NEEDS-SPICE: A System for Easing the Development of Simulation-Ready Compact Models

Tianshi Wang and Jaijeet Roychowdhury

EECS Department
University of California at Berkeley
Outline

• What is a compact model?
  • device equations written for circuit simulators

• How is a compact model used by a simulator?
  → how a simulator sets up equations
  → how it solves the equations
  → how simulation can fail (because of the compact model)
    • how a compact model can help avoid simulation failure

• Flows for developing compact models
  • Berkeley SPICE (direct C code)
  • Verilog-A and commercial simulators
  • NEEDS-SPICE & ModSpec (MATLAB and Verilog-A)
What is a Compact Model

Analysis/Modelling for individual devices ("device simulation")

- Provides detailed information about device operation & characteristics
- Computationally intensive
- EM simulation, drift-diffusion eqns., numerical solution of PDEs, etc.

Compact Model of Device

- Simple enough to be incorporated in circuit simulators
- Accurate enough to have predictive value for circuits
- Terminal behaviour important
  - internal details less important
- Purpose: use in circuit design
  - via circuit simulation
Example: Capacitor

\[ i(t) = \frac{dq}{dt} \]

\[ v(t) \]

Compact Model

\[ q(v) = Cv \]

\[ i(t) = \frac{d}{dt} q(v(t)) \]

\[ C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d} \]

or

\[ C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d} \left[ 1 + \frac{d}{\pi W} + \frac{d}{\pi W} \ln \left( \frac{2\pi W}{d} \right) \right] + \ldots \]

charge capacitance geometrical parameters

electrical parameter

“device level” simulation (eg, finite element electrostatic)

T. Wang and J. Roychowdhury, University of California at Berkeley
Example: Capacitor

Verilog-A Code:

```verilog-a
1 `include "disciplines.vams"
2 `include "constants.vams"
3
4 module Horizontal_Cap(t, b, s);
5
6 inout t, b, s;
7 electrical t, b, s;
8
9 branch (t ,b ) cap;
10 branch (t ,s ) ts;
11 branch (b ,s ) bs;
12
13 parameter real L = 4.0e-6 from (0:inf);
14 parameter real W = 4.0e-6 from (0:inf);
15 parameter real D = 0.6e-6 from (0:inf);
16 parameter real EPSR = 3.9 from (0:inf);
17 // other parameters
18
19 real C, Cpt, Cpb;
20
21 analog begin
22 C = `P_EPS0 * EPSR * L * W / D;
23 Cpt = ...; Cpb = ...;
24 I(cap) <+ ddt(C * V(cap));
25 I(ts) <+ ddt(Cpt * V(ts));
26 I(bs) <+ ddt(Cpb * V(bs));
27 end
28 endmodule
```

Nodes/Branches

- C
- Cp,t
- Cp,b

Parameters

- \[ q(v) = CV \]
- \[ i(t) = \frac{d}{dt} q(v(t)) \]

- \[ C = \frac{\varepsilon_r \varepsilon_0 L \cdot W}{d} \]

Complicated equations for \( C_{p,t} \) and \( C_{p,b} \)

- \[ I_{ts}(t) = C_{p,t} \cdot \frac{dV_{ts}(t)}{dt} \]
- \[ I_{bs}(t) = C_{p,b} \cdot \frac{dV_{bs}(t)}{dt} \]

H. B. Palmer 1927
Example: Capacitor

ModSpec code (MATLAB)

```matlab
1 ... % Defining names, ID, etc
2 vecX = [vts; vbs];
3 vecY = [];
4 vecU = [];
5 ...
6 ... % Defining parameters, branches, etc
7
8 function feout = fe(vecX, vecY, vecU, MOD)
9 feout = zeros(2,1); % [its; ibs]
10 end % fe
11
12 function qeout = qe(vecX, vecY, MOD)
13 ...
14 function fiout = fi(vecX, vecY, vecU, MOD)
15 fiout = [];
16 end % fi
17
18 function qiout = qi(vecX, vecY, MOD)
19 qiout = [];
20 end % qi
```

$$q(v) = Cv$$

$$i(t) = \frac{d}{dt}q(v(t))$$

$$C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d}$$

or

$$C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d} \left[1 + \frac{d}{\pi W} + \frac{d}{\pi W} \ln \left( \frac{2\pi W}{d} \right) \right] + \ldots$$

$$I_{ts}(t) = C_{p,t} \cdot \frac{dV_{ts}(t)}{dt}$$

$$I_{bs}(t) = C_{p,b} \cdot \frac{dV_{bs}(t)}{dt}$$

T. Wang and J. Roychowdhury, University of California at Berkeley

H. B. Palmer 1927

Complicated equations for $C_{p,t}$ and $C_{p,b}$
**Example: MOSFET**

- **Schichman-Hodges (core)**
  \[
  I_{ds} = f(V_{gs}, V_{ds}) = \begin{cases} 
  \beta \left[ (V_{gs} - V_T) - \frac{V_{ds}}{2} \right] V_{ds}, & \text{if } V_{ds} < V_{gs} - V_T \\
  \frac{\beta}{2} (V_{gs} - V_T)^2, & \text{if } V_{ds} \geq V_{gs} - V_T \\
  0, & \text{if } V_{gs} < V_T
  \end{cases}
  \]

- **many other MOS models**
  - BSIM, EKV, PSP, ...
  - MVS
    - MIT Virtual Source compact FET model

---

T. Wang and J. Roychowdhury, University of California at Berkeley
Using Compact Models in Simulation

Simulation Basics Compact Modellers Need to Know
Outline

- What is a compact model?
  - writing device equations in a way useful for circuit design (via circuit simulation)

- What does one do with a compact model?
  - simulation (within a circuit simulator)
    - how a simulator sets up equations
    - how it solves the equations
    - how a compact model can make simulation fail
      - how to work around it when designing a compact model

- Flows for developing compact models
  - Berkeley SPICE (direct C code)
  - Verilog-A and commercial simulators
  - ModSpec (in MATLAB or via Verilog-A)
Circuit Equations

\[ I_{ps} = C \cdot \frac{d}{dt} V_{pn} \]

\[ I_{gs} = 0 \]

Mos: \[ I_{ds} = \begin{cases} f(V_{gs}, V_{ds}) & \text{if } V_{ds} < V_{gs} - V_T \\ \beta \left[ (V_{gs} - V_T) - \frac{V_{ds}}{2} \right] V_{ds} & \text{if } V_{ds} \geq V_{gs} - V_T \\ \frac{\beta}{2} (V_{gs} - V_T)^2 & \text{if } V_{gs} < V_T \end{cases} \]

\[ I_{gs} = 0 \]

Cap: \[ I_{ps} = I \]

Res: \[ I_{ps} = \frac{V_{pn}}{R} \]

Circuit Equations

(made by composing device equations)

KCL\(_1\): \[ \frac{V_{dd} - e_1}{R_1} + C \cdot \frac{d}{dt} (e_1 - e_2) - f_1(V_{in} - e_3, e_1 - e_3) = 0 \]

KCL\(_2\): \[ \frac{V_{dd} - e_2}{R_2} + C \cdot \frac{d}{dt} (e_1 - e_2) - f_2(-e_3, e_2 - e_3) = 0 \]

KCL\(_3\): \[ f_1(V_{in} - e_3, e_1 - e_3) + f_2(-e_3, e_2 - e_3) - I = 0 \]

T. Wang and J. Roychowdhury, University of California at Berkeley
Circuit Equations are DAEs

Differential Equations

Algebraic

DAEs

\[ \frac{d}{dt} \bar{q}(\bar{x}(t)) + \bar{f}(\bar{x}(t), \bar{u}(t)) = \bar{0} \]

\[ \bar{x}(t) = \begin{bmatrix} e_1(t) \\ e_2(t) \\ e_3(t) \end{bmatrix} \]

\[ \bar{q}(\bar{x}) = \begin{bmatrix} -C(e_1 - e_2) \\ +C(e_1 - e_2) \\ 0 \end{bmatrix} \]

\[ \bar{u}(t) = [V_{in}(t)] \]

\[ \bar{f}(\bar{x}, \bar{u}) = \begin{bmatrix} \frac{V_{dd} - e_1}{R_1} - f_1(V_{in} - e_3, e_1 - e_3) \\ \frac{V_{dd} - e_2}{R_2} - f_2(-e_3, e_2 - e_3) \\ f_1(V_{in} - e_3, e_1 - e_3) + f_2(-e_3, e_2 - e_3) \end{bmatrix} \]
Analyses on DAEs: DC/AC/TRAN

DAEs
\[
\frac{d}{dt} \vec{x}(t) + \vec{f}(\vec{x}(t), \vec{u}(t)) = \vec{0}
\]

- DC Analysis
  - solve for \(\vec{x}(t)\) assuming nothing changes with time
    (given constant \(\vec{u}(t)\))

- Transient Analysis
  - solve using Newton-Raphson method

- AC Analysis
  - nonlinear

Vin \equiv 0V

\[\vec{g}(\vec{x}) \triangleq \vec{f}(\vec{x}, \vec{u}) = \vec{0}\]
The Newton Raphson Method

• **Iterative** numerical algorithm to solve \( \vec{g}(\vec{x}) = \vec{0} \)

  → Start with \( \vec{x}_0 \), update \( \vec{x}_i \) with derivative information

\[
\vec{x}_{i+1} = \vec{x}_i - \frac{\vec{g}(\vec{x}_i)}{\frac{d\vec{g}(\vec{x}_i)}{d\vec{x}}}
\]

Start

\[ i = 0 \]

\[ ||\vec{g}(\vec{x}_i)|| \leq \epsilon \]

Yes

No

\[ i \leftarrow i + 1 \]

Done
NR: Convergence

- Conditions for NR to converge reliably
  - $g(x)$ must be "smooth": continuous, differentiable
  - starting guess must be "close enough" to solution
  - if not "close enough", NR is not guaranteed to converge

- However, there are things we can do to improve convergence:
  - initialization and limiting
Convergence Problems in a Simple Circuit

\[ I_D = I_S \left( e^{\frac{e_1}{V_T}} - 1 \right) \]

\[ I_S \approx 1 \times 10^{-12}, V_T \approx 0.025 \]

\[ I_R = \frac{E - e_1}{R} \]

Circuit Equation: \( I_D - I_R = 0 \)

\[ \Rightarrow g(e_1) \triangleq I_S \left( e^{\frac{e_1}{V_T}} - 1 \right) - \frac{E - e_1}{R} = 0 \]

Let \( x = e_1 \), draw \( g(x) \)

What will happen if \( x_0 \) is small?

NR fails for initial guess 0.5

See MATLAB demo

T. Wang and J. Roychowdhury, University of California at Berkeley
NR: Initialization and Limiting

Initialization and limiting are crucial for NR success

- **Initialization**: each “problematic” nonlinear device suggests initial branch voltage/current
  - e.g. diode, \( V_{\text{branch}} \sim 0.7V \)

- **Limiting**: modify NR update step
  - e.g. diode, \( \Delta V_d = \text{pnjlim} (\Delta V_d, V_d) \)

Compact model provides these

See MATLAB demo

```
Start

\( i = 0 \)

\[ \| \vec{g}(\vec{x}_i) \| \leq \epsilon \] Yes

\( \vec{x}_{i+1} = \vec{x}_i - \frac{\vec{g}(\vec{x}_i)}{dg(\vec{x}_i)/d\vec{x}} \)

No

Done

‖g(x_i)‖ ≤ ϵ

“good” \( x_0 \)

compact model should provide hints

Don't use this \( Δx_i \) directly prevent large \( Δx_i \) using limiting

T. Wang and J. Roychowdhury, University of California at Berkeley
Compact Model Development Flows
Outline

• What is a compact model?
  • writing device equations in a way useful for circuit design (via circuit simulation)

• What does one do with a compact model?
  • simulation (within a circuit simulator)
    → how a simulator sets up equations
    → how it solves the equations
    → how a compact model can make simulation fail
      • how to work around it when designing a compact model

• Flows for developing compact models
  • Berkeley SPICE (direct C code)
  • Verilog-A and commercial simulators
  • NEEDS-SPICE & ModSpec (MATLAB, Verilog-A)
SPICE-based Model Development Flow

1. Device Simulation
2. Analytical Equations for Device
3. Write SPICE Model Code (in C)
4. Compile SPICE
5. Run Small Circuits in SPICE
6. Debug code
   - tweak init/limiting
   - check derivatives

   - robust convergence?
     - Yes
     - results OK?
       - Yes
       - Deploy model
       - No
       - Check/fix coding
         - improve equations
         - modify

   - weeks/months

   - not many can do this

   - takes months
Writing SPICE Model Code

- **SPICE**: open source and freely available
- can tweak init/limiting
- Model code is complex
  - many files
- Intricate dependencies
  - easy to break
- Derivatives done by hand
  - error-prone and extremely tedious
- DC/AC/TRAN analysis code inside model!
  - must know SPICE algorithms
  - Typically, takes a PhD student years to deploy compact model
  - needs to learn sim. algorithms
  - needs to learn SPICE coding
    - poorly done; no documentation, ...

- BSIM4
  - total: >20000 lines

- b4ld.c 5122
- b4geo.c 384
- b4getic.c 46
- b4acld.c 670
- b4ask.c 357
- b4.c 1049
- b4check.c 896
- b4cvtest.c 202
- b4del.c 45
- b4dest.c 42

- b4mask.c 2662
- b4mdel.c 50
- b4mpar.c 3619
- b4noi.c 653
- b4par.c 193
- b4pzld.c 757
- b4set.c 2521
- b4temp.c 2324
- b4trunc.c 62
- devsup.c 428
DC/AC/TRAN code in Compact Model!
(SPICE diode code)

```c
#ifndef SENSDEBUG
    printf("vd = %.7e \n", vd);
#endif /* SENSDEBUG */
    goto next1;
}
if(ckt->CKTmode & MODEINITSMSIG) {
    vd = *(ckt->CKTstate0 + here->DIOvoltage);
} else if (ckt->CKTmode & MODEINITTRAN) {
    vd = *(ckt->CKTstate1 + here->DIOvoltage);
} 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) {
```
Verilog-A-based Model Development Flow

1. **Device Simulation**
   - Analytical Equations for Device

2. **Write Verilog-A Model**
   - Takes days or weeks
   - Much easier than coding in SPICE

3. **Run Circuits**
   - in Commercial Simulators
     - Robust convergence?
       - Yes: Results OK?
         - Yes: Deploy model
         - No: Check/fix coding

   - No: Improve equations
     - Trade off physics for convergence
     - Inadequate V-A support in open-source simulators
     - Hand-translated+tweaked “native” implementation needed for efficiency

   - CANNOT TWEAK init/limiting (support obscure/missing)
NEEDS-SPICE Model Development Flow

Device Simulation

Analytical Equations for Device

Write ModSpec Model (in MATLAB)

Test immediately (standalone)

Run Small Circuits in MDE (Model Development Environment)

DC/AC/TRAN in MATLAB

takes hours or days

much more natural than Verilog-A (once learned)

yes robust convergence?

results OK?

No

Run Small Circuits in MDE

Yes

Test immediately (standalone)

debug/improve

add/tweak init/limiting functions

check/fix coding easier/faster in MATLAB

init/limiting → easier physics vs convergence tradeoff

init/limiting → easier physics vs convergence tradeoff

improve equations

satisfactory ModSpec model

No

code available for inspection and debugging

Yes

go to next step in flow

T. Wang and J. Roychowdhury, University of California at Berkeley
NEEDS-SPICE Model Development Flow (2)

**Step 1: model developed in ModSpec/MDE**
- Takes hours or days
- All in MATLAB
- High level of confidence that model works in simulation
  - because you have run DC/AC/TRAN in MDE

**NEEDS-certified Verilog-A model**

**Step 2**
- ModSpec Model (in MATLAB) hand-coded
- ModSpec Model (in MATLAB) auto-generated from Verilog-A
- translate manually to NEEDS-certified Verilog-A
- compare code, check consistency

**end of Step 2: high level of confidence Verilog-A model is correct/debugged**
Step 3

NEEDS-certified Verilog-A model (from Step 2)

Automatic translator

ModSpec Model (C++ API)

compile standalone

already tested in MDE ➔ High probability of success

standardization of NEEDS extensions to Verilog-A

use model in Commercial Simulators

speed near native implementation (compiled C++ code)

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

.model supported in Open-source Simulators

Test immediately (standalone)

fast/efficient

confirm model with DC/AC/TRAN in C++ MDE

standalone proof of model's convergence/speed performance

robust/efficient model deployed

model supported in Open-source Simulators

speed near native implementation (compiled C++ code)
Summary and Conclusion

Analytical Equations for Device

ModSpec model (MATLAB)

ModSpec model (C++)

Verilog-A

Commercial Simulators

NEEDS-SPICE

language issues, no init/limiting, open-source support poor, slow, ....

MDE (DC/AC/TRAN in MATLAB)

MDE (C++)

DC/AC/TRAN

standalone test

init/limiting

open-source simulators

open-source, init/limiting access horrifically complex, time-consuming → IMPRACTICAL

high-level language, no need to know algorithms, MUCH easier than coding in SPICE

NEEDS-SPICE

open-source, init/limiting access

sensible Verilog-A

auto-generated C++

standalone test

reference DC/AC/TRAN

open-source simulators

commercial simulators

fast

→ MATLAB

→ DC/AC/TRAN

→ standalone test

→ init/limiting

→ standalone test

→ fast

→ init/limiting

→ fast