ModSpec: an Open, Universal, Low-Level Model API

Proposal for designation as a CMC-approved standard

Jaijeet Roychowdhury, Tianshi Wang
A. Gokcen Mahmutoglu, Archit Gupta
EECS Department
University of California at Berkeley
What is a low-level model API?

- **Illustrative example: diode (incl. junction charges)**

  ![Diode Diagram]

  \[ i = I_s \left( e^{v/V_t} - 1 \right) \]

  - a **complicated piecewise expression**

  » a low-level model API needs to provide

  - an **executable function** for i-v characteristics: eg, \( i = g(v) \)
    - for which, need to know device's IOs
  - a function for q-v characteristics
  - (for multiple terminals – any number)
  - names, default values, descriptions of parameters…
  - facilities for incorporating internal nodes/unknowns…
  - derivatives, bookkeeping information, …
Verilog-A is NOT a low-level model API

- Eg, no well-defined **executable** functions as API
  - VA is a netlist-like model **description** language
- VA models **require translation** to run
  - translated code is **not standalone**
    - has to be compiled into and run within a simulator
      - simulators can **interpret same VA constructs differently**
    - the code is typically simulator dependent (eg, SPICE)
      - has to be **aware of simulator internals** (data structures, functions, …)
      - eg, ADMS needs a **customized backend for every simulator**
    - same VA code translatable to **different** low-level models
      - which behave differently (due to finite-precision numerics)
Standardized low-level model API: Benefits

- Enables **precise** definition of model
  - independent of simulator or analysis algorithm
  - exact same model code portable to every simulator
    - including compiled binary versions (e.g., for IP protection)
    - promotes consistency of model across simulators
  - great aid to **model development and debugging**
    - key: model is **executable and testable without a simulator**
      - separates model from implementation-in-simulator → model bugs and simulator-implementation related bugs isolated
      - debug model standalone – including in MATLAB (convenient, accessible to device physics community)
      - promotes development of "simulation-ready" models
    - models that don't work: the bane of every simulator engineer's life
ModSpec: a low-level device API

• **Based on a solid mathematical foundation**
  » not just EE devices; authentically multi-physics
  » devices have IOs, internal unknowns, diff. eqns., …
    – (details: see Formulating ModSpec slides sent separately)

• **Open, transparent; examples + (useful) docs**

• **Precise API independent of simulators/analyses**
  » executable code for “just the model”
  » same API in MATLAB and C/C++

• **Supporting toolchains (all open and transparent)**
  » VAPP: Verilog-A to ModSpec translator
  » MAPP/Xyce: simulators that take ModSpec models
    - MAPP: `check_ModSpec()`, `model_exerciser()` utilities
VAPP: Verilog-A Parser and Processor

- Verilog-A device model
- Verilog-A Parser and Processor (VAPP)
- executable and debuggable standalone (no simulator needed)
- easy to examine or write by hand
- mathematically well defined, modular
- supports every analysis (DC, tran, AC, PSS, ...) without device modeller having to know anything about them
- general: any device in any physical domain
- easily and directly usable by any simulator
Device Model Development Using ModSpec

**NEEDS**
- compatible Verilog-A

**Compact Model Equations**

**ModSpec (MATLAB version)**

**Verilog-A Parser and Processor**
- VAPP
- format itself eliminates some common modelling mistakes

**Test immediately (standalone)**

**Run Small Circuits in MAPP**

**Problems?**
- No
- Yes

- model doesn't evaluate
- overflow/domain
- DC conv. failure
- transient timestep too small
- unphysical results
- voltage/current blows up
- smoothing
- define custom functions
- custom init/limiting
- gmin

**DC/AC/TRAN/etc in MATLAB**

- code/facilities for inspection and debugging

- format itself eliminates some common modelling mistakes
Model Development Flow (2)

Step 2

NEEDS-compatible Verilog-A model

use model in Commercial Simulators

Already tested in MAPP → High probability of success

NEEDS-compatible Verilog-A model

VAPP C++ backend

ModSpec Model (C++ API)

compile standalone

.so libraries (dynamically loadable)

Test immediately (standalone)

fast/efficient

speed near native implementation (compiled C++ code)

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

model supported in Open-source Simulators (Xyce)

binary release possible (IP protection)

standalone proof of model's convergence/speed performance

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

“simulation ready” model deployed

J. Roychowdhury, Berkeley
Examples

VAPP comes with a suite of examples. Each example consists of a different model and a MAPP script to perform an analysis using that model. The following models are available within the examples.

- BSIM-3
- BSIM-4
- BSIM-6
- MVS-1
- PSP
- Purdue Negative Capacitance FET
- R3
- VBIC

- Ring oscillator
- Characteristic curves
- CMOS inverter
- CS Amplifier
- Homotopy
- Transient with self-heating
- Differential pair
VAPP – Gallery of Models

Selected Models translated with VAPP

BSIM-3, BSIM-4, BSIM-BULK, PSP, MVS, VBIC, R3
% give these commands in MATLAB with MAPP and VAPP loaded
va2modspec('mvs_si_1_1_0_vappmod.va');
MOD = mvs_si_1_1_0();
MOD = MOD.setparms('parm_Rs0', 1e10, MOD);
MEO = model_exerciser(MOD); MEO.display(MEO)

vdb=0; vgb=-2:0.1:2; vsb=0; vdisi=-2:0.05:2; vbsi=0;
MEO.plot('KCLForI_disi_fi', vdb, vgb, vsb, vdisi, vbsi, MEO)
MEO.plot('dKCLForI_disi_fi_dv_disi', vdb, vgb, vsb, vdisi, vbsi, MEO); % discontinuity at vdisi=0
vdb=0; vgb=-2:0.1:2; vsb=0; vdisi=-1.2; vbsi=0;
MEO.dKCLForI_disi_fi_dvgb(vdb, vgb, vsb, vdisi, vbsi, MEO) % shows NaNs
ModSpec vs Existing Model Interfaces

- ModSpec is **completely open** (and open source)
  - openness key for preventing Balkanization of standards
- ModSpec is **clean, general, easy to comprehend**

- 2n IOs, split into:
  - l explicit outputs (0 <= l <= n): \( \vec{x} \)
  - 2n-l “otherIOs”: \( \vec{y} \)
  - m internal unknowns: \( \vec{u} \)
- \( m \) independent sources: \( \vec{u}(t) \)
- \( l \) equations for the explicit outputs
  \[ \dot{\vec{x}}(\vec{x}, \vec{y}, \vec{u}) = \frac{d}{dt} \vec{q} (\vec{x}(t), \vec{y}(t), \vec{u}(t)) + \vec{f}(\vec{x}(t), \vec{y}(t), \vec{u}(t)) \]
- \( n+m-l \) remaining (implicit) equations
  \[ \ddot{\vec{x}}(\vec{x}, \vec{y}, \vec{u}) = \frac{d}{dt} \ddot{\vec{q}} (\vec{x}(t), \vec{y}(t), \vec{u}(t)) + \ddot{\vec{f}}(\vec{x}(t), \vec{y}(t), \vec{u}(t)) = 0 \]

CMC_TMI2 evolved from SPICE model structuring
Glimpse: ModSpec-C++ in Xyce

1 *** Test-bench for generating dc response of an inverter
2 *** Create sub-circuit for the inverter
3 .subckt inverter Vin Vout Vvdd Vgnd
4
5 yModSpec_Device X1 Vvdd Vin Vout Vvdd[MVSmod; type=-1 W=1.0e-4
6 Lgdr=32e-7 dLg=8e-7 Cg=2.57e-6 beta=1.8 alpha=3.5 Tjun=300
7 Cif = 1.38e-12 Cof=1.47e-12 phib=1.2 gamma=0.1 mc=0.2
8 CTM_select=1 Rs0=100 Rd0 = 100 n0=1.68 nd=0.1 vxo=7542204
9 mu=165 Vt0=0.5535 delta=0.15
10
11 yModSpec_Device X0 Vout Vin Vgnd Vgnd MVSmod; type=1 W=1e-4
12 Lgdr=32e-7 dLg=9e-7 Cg=2.57e-6 beta=1.8 alpha=3.5 Tjun=300
13 Cif=1.38e-12 Cof=1.47e-12 phib=1.2 gamma=0.1 mc=0.2
14 CTM_select=1 Rs0=100 Rd0=100 n0=1.68 nd=0.1 vxo=1.2e7
15 mu=200 Vt0=0.4 delta=0.15
16
17 .model MVSmod MODSPEC_DEVICE SONAME=MVS_ModSpec_Element.so
18
19 .ends
20
21 *** circuit layout
22 Vsup sup 0 1
23 Vin in 0 0
24 Vsource source 0 0
25 X2 in out sup 0 inverter
26
27 *** simulation
28 .dc Vin 0 1 0.01
29
30 .print dc V(in) V(out)
31
32 *** END
33 .end

J. Roychowdhury, Berkeley
ModSpec: designed for Multi-Physics

\[\begin{align*}
\dot{z}(\vec{x}, \vec{y}, \vec{u}) &= \frac{d}{dt}\left[\dot{q}_e(\vec{x}(t), \vec{y}(t), \vec{u}(t)) + \dot{f}_e(\vec{x}(t), \vec{y}(t), \vec{u}(t))\right] \\
\dot{w}(\vec{x}, \vec{y}, \vec{u}) &= \frac{d}{dt}\left[\dot{q}_i(\vec{x}(t), \vec{y}(t), \vec{u}(t)) + \dot{f}_i(\vec{x}(t), \vec{y}(t), \vec{u}(t))\right] = 0
\end{align*}\]

general; define model completely; executable/debuggable standalone; easily used by simulator algorithms

general multi-physics support
ModSpec/MAPP: Multiphysics Capabilities

Electro-Thermal

Modelling Devices with Hysteresis

Spintronic

Bio-chemical

RRAM/memristor

ESD snapback
Thermistor

The resistance of a thermistor varies with temperature.

\[ \text{ipn} = \frac{\text{vpn}}{R}, \]

where \( R = R_0 + k \cdot (\text{temp}_t - T_0) \).

\[ \text{pwr}_t = -\text{ipn} \cdot \text{vpn}, \]

Electrical nodes and thermal nodes

**ModSpec Core**

**I/Os:**
- **ipn**, \( vpn 
- **pwr\_t**, \( \text{temp}_t \)

**Equations:**
\[ \text{pwr}_t = -\text{ipn} \cdot \text{vpn}; \]
\[ 0 = \text{ipn} - \frac{\text{vpn}}{R_0 + k \cdot (\text{temp}_t - T_0)}. \]

**I/O Name** | **I/O Type** | **I/O Nodes**
---|---|---
**ipn** | current | \( p, n \)
**vpn** | voltage | \( p, n \)

**thermalNIL**

<table>
<thead>
<tr>
<th>I/O Name</th>
<th>I/O Type</th>
<th>I/O Nodes</th>
</tr>
</thead>
</table>
| **pwr\_t** | power flow | \( t, \text{abs}_\text{zero} \)
| **temp\_t** | temperature | \( t, \text{abs}_\text{zero} \)

**Data Structure Describing I/Os**

**fe:** -ipn*vpn, \( \text{qe: } 0 \)
**fi:** -ipn-vpn/R(temp\_t), \( \text{qi: } 0 \)
Thermistor

**Thermal KVL:**
Each node has a temperature

**Thermal KCL:**
Power flows sum up to zero

---

**Electrical Network**

- $V_1$
- $R_1$
- $C_{th}$
- $V_{amb}$

**Thermal Network**

- $n_1$
- $t_{n1}$
- $R_{th}$
- $t_{namb}$

---

**Netlist**

```plaintext
.netlist
  .name: (string) 'Vsrc-thermistor'
  .nodenames: (hash table) 'electrical' {'n1'}
  'thermal' {'tn1', 'tnamb'}
  .domain_add_ons: (hash table)
  'electrical' .groundnodename='gnd'
  'thermal' .abszeronodename='abs_zero'
  .elements: (cellarray of structs)
  .output: (cellarray of structs)
```

---

**Graph**

- vsrcthermistor: transient using GEAR2 LMS solver

- Values vs. time (s)
- Time range: 0 to 0.5 s
- Y-axis range: -100 to 350
Multiphysics Systems

potential/flow systems:

kinematic NIL:
“flow”: force
“potential”: position

magnetic NIL:
“flow”: magnetic flux
“potential”: magnetomotive force

thermal NIL:
“flow”: power flow
“potential”: temperature

Spintronic systems:

vectorized spin currents
vectorized spin voltages

Optical systems

Chemical reaction networks

rates and concentrations
“KCLs” at nodes have d/dt terms

Kerem Yunus Camsari; Samiran Ganguly; Supriyo Datta (2013), "Modular Spintronics Library," https://nanohub.org/resources/17831.
Devices with Hysteresis

memristor
(RRAM, CBRAM, PCM...)

ESD snapback

magnetic core

How to model hysteresis?

```
1) real i;
2) analog begin
3)   if V(br) < -1
4)     i = -1;
5)   if V(br) > +1
6)     i = +1;
7)   I(br) <+ i;
8) end
```

```
@initial_step)
begin
   x = x_init;
end
```

```
$bound_step(tstep);
c_time = $abstime;
dt = c_time - p_time;
x = x_last + dt * exp(...);
```

Boolean variable
“hybrid model”

```
1) int isON = 0;
2) if (abs(V(...)) > V_snap)
3)   isON = 1;
4) if (isON) {
5)     ...
6) } else {
7)     ...
8) }
```

Only for TRAN
None works for DC, AC, PSS

**memristor**
(RRAM, CBRAM, PCM...)

- Linear/nonlinear ion drift models
  Biolek (2009), Jogelkar (2009), Prodromakis (2011), etc.
- UMich RRAM model (2011)
- TEAM model (2012)
- Simmons tunneling barrier model (2013)
- Yakopcic model (2013)
- Stanford/ASU RRAM model (2014)
- Knowm “probabilistic” model (2015)

Not an analog compact model

“memory state”
“hidden state”

Not an analog
Verilog-A problems

failure in simulation

problematic physics

poor understanding of VA

poor knowledge of the mathematics of computational modelling
How to Model Hysteresis Properly

\[ \text{ipn} = f(\text{vpn}) \]

\[ \text{ipn} = f_1(\text{vpn}, s) \]

\[ \frac{d}{dt}s = f_2(\text{vpn}, s) \]

**Example:**

\[ f_1(\text{vpn}, s) = \frac{\text{vpn}}{R} \cdot (1 + \tanh(s)) \]

\[ f_2(\text{vpn}, s) = \text{vpn} - s^3 + s \]

internal state variable “memory”
How to Model Hysteresis Properly

**Template:**
\[ ipn = f_1(vpn, s) \]
\[ \frac{d}{dt}s = f_2(vpn, s) \]

**ModSpec:**
\[ ipn = \frac{d}{dt}q_e(vpn, s) + f_e(vpn, s) \]
\[ 0 = \frac{d}{dt}q_i(vpn, s) + f_i(vpn, s) \]
How to Model Hysteresis Properly

all DC sols from homotopy analysis (like a curve tracer)
ESD Snapback Model

ESD protection device


\[ I_{\text{on}} = G_{\text{on}} \cdot (V - V_H) \]
\[ I_{\text{off}} = I_S \cdot (1 - e^{-V/\phi_T}) \cdot \sqrt{1 + \frac{\max(V, 0)}{V_D}} \]
\[ I = s \cdot I_{\text{on}} + I_{\text{off}} \]

internal state: indicator of impact ionization

\[ \frac{d}{dt}s = f(V, s) \]

J. Roychowdhury, Berkeley
ESD Snapback Model

- **forward/backward DC sweeps**
- **homotopy (all DC sols)**
- **transient voltage sweeps**

“impact ionization doesn't happen instantaneously (although all compact models assume that it does)” — C.C. McAndrew

Human Body Mode (HBM) test

![ESD clamp diagram](image)
ESD Snapback Model

Simple TLP equivalent circuit

MM

CDM

second snapback
Memristor Models

\[ \frac{d}{dt} s = f_2(vpn, s) \]

Available \( f_2 \):

1. **linear ion drift**
   \[ f_2 = \mu_v \cdot R_{on} \cdot f_1(vpn, s) \]

2. **nonlinear ion drift**
   \[ f_2 = a \cdot vpn^m \]

3. **Simmons tunnelling barrier**
   \[ f_2 = \begin{cases} c_{off} \cdot \sinh\left( \frac{1}{t_{off}} \right) \cdot \exp\left( -\exp\left( \frac{s - a_{off}}{w_c} - \frac{1}{b} \right) - \frac{s}{w_c} \right), & \text{if } i \geq 0 \\ c_{on} \cdot \sinh\left( \frac{1}{t_{on}} \right) \cdot \exp\left( -\exp\left( \frac{a_{on} - s}{w_c} + \frac{1}{b} \right) - \frac{s}{w_c} \right), & \text{otherwise,} \end{cases} \]

4. **TEAM model**

5. **Yakopcic model**

6. **Stanford/ASU**
   \[ f_2 = -v_0 \cdot \exp\left( -\frac{E_a}{V_T} \right) \cdot \sinh\left( \frac{vpn \cdot \gamma \cdot a_0}{t_{ox} \cdot V_T} \right) \]

Available \( f_1 \):

1. \[ f_1 = (R_{on} \cdot s + R_{off} \cdot (1 - s))^{-1} \cdot vpn \]

2. \[ f_1 = \frac{1}{R_{on}} \cdot e^{-\lambda \cdot (1-s)} \cdot vpn \]

3. \[ f_1 = s^n \cdot \beta \cdot \sinh(\alpha \cdot vpn) + \chi \cdot (\exp(\gamma) - 1) \]

4. \[ f_1 = \begin{cases} A_1 \cdot s \cdot \sinh(B \cdot vpn), & \text{if } vpn \geq 0 \\ A_2 \cdot s \cdot \sinh(B \cdot vpn), & \text{otherwise.} \end{cases} \]

5. \[ f_1 = I_0 \cdot e^{-\text{Gap}/g_0} \cdot \sinh\left( \frac{vpn}{V_0} \right) \]
   \[ \text{Gap} = s \cdot \text{minGap} + (1 - s) \cdot \text{maxGap}. \]

- set up boundary
- fix \( f_2 \) flat regions
- smooth, safe funcs, scaling, init/limiting, etc.
Memristor Models

A collection of 30 models:
- all smooth, all well posed
- not just RRAM, but general memristive devices
- not just bipolar, but unipolar
- not just DC, AC, TRAN, but homotopy, PSS, ...

PSS using HB
STEAM
“Spline-based Tables for Efficient and Accurate Device-Modeling”

Archit Gupta, Tianshi Wang, Ahmet Gokcen Mahmutoglu, Jaijeet Roychowdhury
EECS Department, UC Berkeley
Revisiting table-based device modeling
Key questions

- What should these tables look like for a generic compact model (Not specifically MOSFETs, BJTs etc.)
- How are the device dynamics obtained from these tables?
From ModSpec to table-based model

MODSPEC

Machine Translation from Verilog-A is openly available – VAPP

32 nm Planar Transistors

4n SPLINE COEFFICIENTS
Speedup Evaluation

Core device evaluation
BSIM 100-150X, MVS 20-40X

Analysis (BSIM)
DC (QSS) 14-20X, AC 7-10X, TRANSIENT 6-8X
Error Evaluation

Core device evaluation
BSIM < 0.00001%
MVS < 0.001%

Analysis
DC < 0.1%,
AC < 0.1%,
TRANSIENT < 0.01%
ModSpec/VAPP/MAPP: Contributors

Special thanks to:

Mark Lundstrom (Purdue)

Eric Keiter
Sandia Labs

Colin McAndrew
Freescale/NXP

Geoffrey Coram
Analog Devices

Peter Bermel
Purdue

Gokcen Mahmutoglu

Archit Gupta

Jaijeet Roychowdhury

Tianshi Wang

Kiran Gullapalli
Freescale/NXP

Guofu Niu
Auburn

Tom Russo
Sandia Labs

Karthik Aadithya
Sandia Labs

Xufeng Wang
Purdue

Jason Verley
Sandia Labs