MAPP: The Berkeley Model and Algorithm Prototyping Platform

Tianshi Wang, Aadithya V. Karthik, Bichen Wu, Jian Yao and Jaijeet Roychowdhury
EECS Department, University of California, Berkeley

***Abstract*—We present the Berkeley Model and Algorithm Prototyping Platform (MAPP), a MATLAB®-based framework for conveniently and quickly prototyping device compact models and simulation algorithms. MAPP's internal code structuring, which differs markedly from that of Berkeley SPICE and related simulators, allows users to add new devices with only minimal knowledge of simulation algorithms, and vice-versa. We describe MAPP's structuring and provide an overview of its capabilities. MAPP is available as open source under the GNU Public License.**

I. Introduction

A long-standing barrier to research in device models and simulation algorithms has been the lack of a powerful yet easy-to-use platform for prototyping new ideas. While Berkeley SPICE [1], [2] and its derivatives, *e.g.*, ngspice [3], have long been standard platforms for modelling and simulation, they are not well suited for quick and convenient prototyping, primarily because of outdated code structuring. In particular, simulation algorithms in Berkeley SPICE are implemented in, and specialised for, every device model. Such structuring makes it difficult to insert new devices or algorithms. While modern open-source post-SPICE circuit simulators, *e.g.*, Gnucap [4] and Qucs [5], have many advantages over SPICE and its derivatives, especially in code readability and documentation, they continue to implement algorithms in devices, hence adding either remains challenging. Xyce [6], a modern open source simulator, allows device models to be nearly independent of analysis types, alleviating the difficulty of developing models and algorithms to a great extent. However, code development in Xyce requires considerable facility with C++ programming.

This situation has long hindered research in modelling and simulation – the barrier to entry for incorporating new devices or analyses is so high that few researchers are capable of performing these tasks effectively. New ideas are often dropped simply because they cannot be prototyped in a reasonable time using available open source simulators.

To address this issue, we have been developing the Berkeley Model and Algorithm Prototyping Platform (MAPP). The primary goal of MAPP is to ease the process of developing new device models and simulation algorithms, especially for those who do not have an extensive background in compact modelling or experience coding algorithms in simulators. Towards this end, we have chosen to implement MAPP entirely in MATLAB®. MATLAB® is widely used today in scientific and engineering communities; its simple, mathematics-based syntax makes programming accessible to a broad range of users. In addition, MATLAB® is interactive, as well as interpreted (*i.e.*, there is no need for compilation), which makes it well suited for quick prototyping and debugging. It has built-in support for vectors/matrices (including sparse matrices) and comes with an exceptionally rich set of mathematical objects and functions in linear algebra, statistics, Fourier analysis, optimization, bioinformatics, *etc.*, all useful for compact modelling and simulation. MATLAB® also offers flexible, easy-to-use graphics/visualization facilities which are valuable when exploring new devices and analyses.

MAPP's code structuring, which differs markedly from that of SPICE, makes prototyping models and algorithms fast and easy. As shown in Fig. 1, the structure of MAPP centers around a mathematical abstraction, the Differential Algebraic Equation (DAE) [7], which is well suited for describing continuous-time dynamical systems in virtually any physical domain. The use of DAEs enables MAPP to model and simulate devices and systems from domains beyond just the electrical in a natural way. Device equations are specified in a MATLAB®-based format (ModSpec [8]). With the DAE concept separating device equations and analysis algorithms, MAPP's ModSpec device models are unaware of what simulation algorithms there are that may use them. This simplifies and speeds up the task of device model prototyping in MAPP.

DAEs are set up by MAPP's Equation Engines, which combine network connectivity information (*e.g.*, from a circuit netlist) with device model equations in ModSpec to produce system-level DAEs. MAPP's simulation algorithms are aware only of DAE objects; they work by calling DAE accessor functions (collectively called DAEAPI). This structuring enables developers to add new algorithms knowing only the generic format of DAEs, without having to look into the details of device implementation or equation formulation.

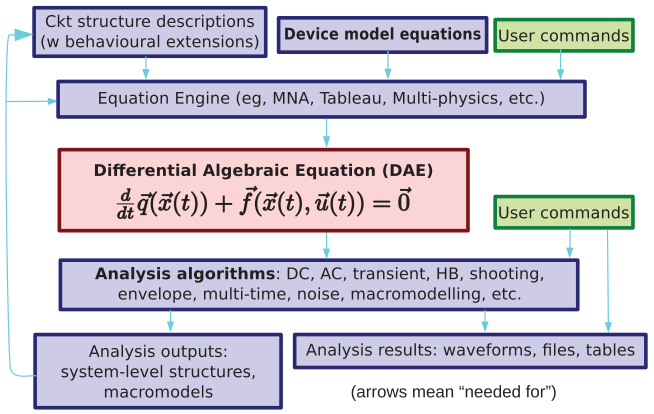

Ckt structure descriptions (w behavioural extensions)
Device model equations
User commands
Equation Engine (eg, MNA, Tableau, Multi-physics, etc.)
Differential Algebraic Equation (DAE)
$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t), \vec{u}(t)) = \vec{0}$
User commands
Analysis algorithms: DC, AC, transient, HB, shooting, envelope, multi-time, noise, macromodelling, etc.
Analysis outputs: system-level structures, macromodels
Analysis results: waveforms, files, tables
(arrows mean "needed for")

Fig. 1. Components of MAPP.

For several years before its public release, MAPP has been used internally in our group and has greatly facilitated our own research. For instance, we developed, implemented and validated a new algorithm for distortion computation [9] using MAPP; this was done by a fresh graduate student, new to the field, in about three weeks. By way of comparison, the much more limited distortion capability in Berkeley SPICE3 [2] had taken over a year to implement — by a graduate student already familiar with coding within SPICE3. Implementing our new algorithm would have involved making so many changes to SPICE3, and taken so much time and effort, that we would not even have tried; the idea would have been lost. Not only has MAPP been useful for trying new ideas, it has also served as a key vehicle for meaningful collaboration with other research groups. Furthermore, it has been helpful for teaching simulation and modelling concepts, with preliminary versions used in classes over several years.

MAPP has been released publicly in open source form [10], primarily under the GNU Public License, with alternative licensing models also supported. The current release of MAPP contains common electrical devices and standard simulation algorithms, including DC, AC and transient analyses. Many more capabilities, including multi-physics device and system modelling features, and additional analyses such as shooting, harmonic balance, homotopy, stationary noise analysis, parameter sensitivity analysis, per-element distortion analysis, model order reduction based on moment matching and Krylov subspace methods, *etc.* [7], have already been prototyped in MAPP and will also be made available under MAPP's open source license. Users can contribute code to MAPP via a public git repository. MAPP comes with a suite of examples at the device, system and analysis levels. It leverages MATLAB®'s help system to help new users get started, and to provide more advanced users information about internal structuring and available functions. It also includes an automatic testing system, designed to facilitate development by quickly detecting problems as code is written or changed. All these features help make MAPP ideal for rapid prototyping of device models and simulation algorithms.

The rest of the paper is organized as follows. In Sec. II, we illustrate the flow of device compact modelling in MAPP. In Sec. III, we discuss features that make algorithm prototyping quick and convenient in MAPP. We then present several examples of models/algorithms in MAPP in Sec. IV.

II. Device Compact Modelling in MAPP

Fig. 2 depicts the device model prototyping flow in MAPP. We use a tunnel diode example below to explain the steps that constitute this flow, illustrating its advantages and novelties along the way.

<u>Step 1.1: *Writing the model in ModSpec*</u>:

Writing a model will normally start by specifying model equations in the ModSpec format [8], which we illustrate using a tunnel diode model. Tunnel diodes [11] are a type of two-terminal semiconductor device with I/V characteristics similar to the blue curve in Fig. 3. We name the model's two terminals p and n, as shown in Fig. 4. Associated with these terminals are two electrical I/O properties: a branch voltage vpn and a branch current ipn. In their simplest form, their relationship can be written as [11]

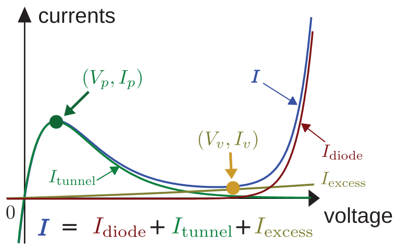


Fig. 3. Tunnel diode I/V curve.

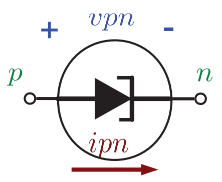


Fig. 4. Tunnel diode schematic.

$$i_{pn} = \frac{d}{dt}(C \cdot v_{pn}) + I_{\text{diode}} + I_{\text{tunnel}} + I_{\text{excess}}, \tag{1}$$

where $C \cdot v_{pn}$ models the charge between the terminals. I_{diode}, I_{tunnel} and I_{excess} are the regular diode current, tunnelling current and additional parasitic tunnelling current terms, respectively:

$$I_{\text{diode}} = I_s \cdot \exp(\frac{v_{pn}}{V_t} - 1), \tag{2}$$

$$I_{\text{tunnel}} = \frac{I_p}{V_p} \cdot v_{pn} \cdot \exp(-\frac{v_{pn} - V_p}{V_p}), \tag{3}$$

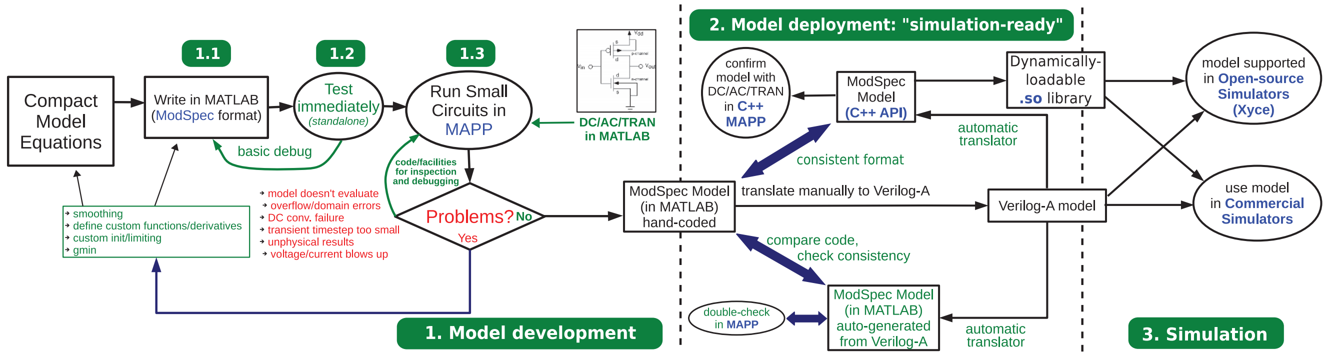
$$I_{\text{excess}} = \frac{I_v}{V_v} \cdot v_{pn} \cdot \exp(v_{pn} - V_v). \tag{4}$$

The above equations involve the model parameters C, I_s, V_t, V_p, I_v and V_v, which determine the shape of the tunnel diode's characteristic curve and its dynamics. (1) is a nonlinear differential equation, with one of its I/Os, namely ipn, expressed explicitly – such differential equations are at the core of all device compact models. ModSpec supports the following general system of equations for devices [8]:

$$\vec{z} = \frac{d}{dt}\vec{q}_e(\vec{x}, \vec{y}) + \vec{f}_e(\vec{x}, \vec{y}, \vec{u}), \tag{5}$$

$$0 = \frac{d}{dt}\vec{q}_i(\vec{x}, \vec{y}) + \vec{f}_i(\vec{x}, \vec{y}, \vec{u}). \tag{6}$$

The vector quantities $\vec{x}$ and $\vec{z}$ contain the device's terminal I/Os: $\vec{z}$ comprises those I/Os that can be expressed explicitly (ipn for our tunnel diode example), while $\vec{x}$ comprises those that cannot (vpn for the tunnel diode). $\vec{y}$ contains the model's internal unknowns (*e.g.*, internal nodes), while $\vec{u}$ provides a mechanism for specifying time-varying inputs within the device (*e.g.*, as in independent voltage or current sources). (Our tunnel diode example has no entries in $\vec{y}$ and $\vec{u}$.) The functions $\vec{q}_e$, $\vec{f}_e$, $\vec{q}_i$ and $\vec{f}_i$ define the differential and algebraic



1.1
1.2
1.3
Compact Model Equations
Write in MATLAB (ModSpec format)
Test immediately (standalone)
Run Small Circuits in MAPP
DC/AC/TRAN in MATLAB
basic debug
code/facilities for inspection and debugging
smoothing
define custom functions/derivatives
custom init/limiting
gmin
model doesn't evaluate
overflow/domain errors
DC conv. failure
transient timestep too small
unphysical results
voltage/current blows up
Problems?
No
Yes
1. Model development
2. Model deployment: "simulation-ready"
confirm model with DC/AC/TRAN in C++ MAPP
ModSpec Model (C++ API)
Dynamically-loadable .so library
automatic translator
consistent format
ModSpec Model (in MATLAB) hand-coded
translate manually to Verilog-A
Verilog-A model
compare code, check consistency
double-check in MAPP
ModSpec Model (in MATLAB) auto-generated from Verilog-A
automatic translator
model supported in Open-source Simulators (Xyce)
use model in Commercial Simulators
3. Simulation

Fig. 2. Device model prototyping flow in MAPP.

parts of the model's explicit and implicit equations. The tunnel diode (1) can be expressed in ModSpec as

$$\begin{aligned}
&\vec{f}e(\vec{x},\vec{y},\vec{u}) = I_{\text{diode}}(\vec{x}) + I_{\text{tunnel}}(\vec{x}) + I_{\text{excess}}(\vec{x}),\\
&\vec{q}e(\vec{x},\vec{y}) = C\cdot\vec{x}, \qquad\qquad (7)\\
&\vec{f}i(\vec{x},\vec{y},\vec{u}) = [\,],\quad \vec{q}i(\vec{x},\vec{y}) = [\,],
\end{aligned}$$

with $\vec{x} = [vpn]$, $\vec{y} = [\,]$, $\vec{z} = [ipn]$, $\vec{u} = [\,]$. Issuing the command "`help ModSpec_concepts`" within MAPP provides more detailed explanations of these concepts.

ModSpec objects in MAPP are simply MATLAB® structures that contain a number of datum and function-handle fields. "`help ModSpecAPI`" in MAPP provides detailed documentation of these fields. Writing a ModSpec device model involves providing basic model information (*e.g.*, the number of terminals, internal nodes, which I/Os are explicitly available, parameter names/values, *etc.*) as data fields, and writing the model functions $\vec{q}_e$, $\vec{f}_e$, $\vec{q}_i$, $\vec{f}_i$ using standard MATLAB® syntax. For example, the tunnel diode model above is described with the following ModSpec code:

```
function MOD = Tunnel_Diode_ModSpec()
  MOD = ee_model();
  MOD = add_to_ee_model(MOD,'terminals', {'p', 'n'});
  MOD = add_to_ee_model(MOD,'explicit_outs', {'ipn'});
  MOD = add_to_ee_model(MOD,'parm',{'Is',1e-12,'Vt',0.025});
  MOD = add_to_ee_model(MOD,'parm',{'Ip',3e-5,'Vp',0.05});
  MOD = add_to_ee_model(MOD,'parm',{'Iv',3e-6,'Vv',0.3});
  MOD = add_to_ee_model(MOD,'parm',{'C', 1e-15});
  MOD = add_to_ee_model(MOD,'fe', @fe);
  MOD = add_to_ee_model(MOD,'qe', @qe);
  MOD = finish_ee_model(MOD);
end

function out = fe(S)
  v2struct(S);
  I_diode = Is*(exp(vpn/Vt)-1);
  I_tunnel = (Ip/Vp) * vpn * exp(-1/Vp * (vpn - Vp));
  I_excess = (Iv/Vv) * vpn * exp(vpn - Vv);
  out = I_diode + I_tunnel + I_excess;
end

function out = qe(S)
  v2struct(S);
  out = C*vpn;
end
```

In Fig. 5, an excerpt from SPICE3's implementation of a regular diode model is shown as comparison. The code shown has almost no direct relation to the diode's equations; it relates to the implementations of AC, transient, *etc.*, analyses in SPICE3 — in which the complete diode implementation involves 27 different files, with 2704 lines of code in all. In contrast, the 25 lines of ModSpec code above describe the entire tunnel diode model; it works in every analysis in MAPP.

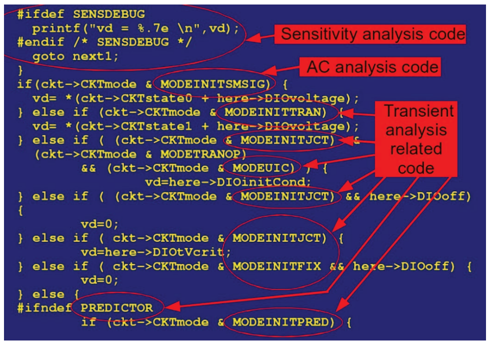

```
#ifdef SENSDEBUG
  printf("vd = %.7e \n",vd);
#endif /* SENSDEBUG */
  goto next1;
}
if(ckt->CKTmode & MODEINITSMSIG) {
  vd= *(ckt->CKTstate0 + here->DIOvoltage);
} else if (ckt->CKTmode & MODEINITTRAN) {
  vd= *(ckt->CKTstate1 + here->DIOvoltage);
} else if ( (ckt->CKTmode & MODEINITJCT) &
  (ckt->CKTmode & 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) {
```

Fig. 5. Excerpt from SPICE3's dioload.c, illustrating how every device contains code for every analysis.

Although the tunnel diode example here is a simple two-terminal device, the ModSpec format supports more general devices, with multiple terminals, internal nodes, *etc.*, through its vector equations (5) and (6). ModSpec also supports specifying parameters, noise sources and other features (such as device-specific limiting and initialization) [8], [7]. Since the format itself makes modellers explicitly aware of important mathematical features of the model (such as the numbers of equations and unknowns involved, which equations are purely algebraic and which involve differential terms, *etc.*), many common modelling errors are eliminated.

Another implication of the differential equation format (5) and (6) is that ModSpec devices are not limited to any specific physical domain. Domain-specific attributes (*e.g.*, voltage/current concepts for electrical devices, together with related constraints such as KCL and KVL [7]) are layered on through a Network Interface Layer (NIL), an add-on structure within ModSpec. Specifying several NILs for a single ModSpec device makes it easy to model multi-physics devices, as illustrated in Fig. 6. Perhaps most importantly, writing a device model in ModSpec enables immediate, standalone testing and model debugging in MATLAB®, without necessarily relying on any of MAPP's analyses.

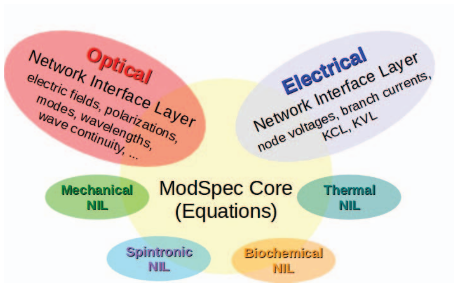


Fig. 6. ModSpec supports multiple physical domains in the same device through the concept of the Network Interface Layer (NIL).

Step 1.2: *Testing the model standalone*: The ModSpec format is a MATLAB® structure and contains executable function fields. It allows modellers to evaluate and visualize the model's functions right after it is coded in MAPP, without incorporation within a circuit. This is useful for checking equation correctness and catching simple bugs at an early stage of the model development flow. The functions tested at this point are the same ones called during circuit simulation; since no translation or interpretation is involved, model development and deployment more transparent and reliable. Existing Verilog-A based model development flows and tools do not provide a standalone check capability, since they necessarily involve translation/interpretation implemented in each simulator; the only way such a model can be exercised by the user is by writing test circuits and running them in the simulator.

Continuing with the tunnel diode example, by evaluating $\vec{f}_e$ with different input voltages $\vec{x} = vpn$, we can plot the I/V curve of the tunnel diode (shown in Fig. 7) without putting it in a circuit. The ModSpec model also contains automatically-generated functions for the derivatives of $\vec{q}_e$, $\vec{f}_e$, $\vec{q}_i$, $\vec{f}_i$, *etc.*; the derivatives are computed by MAPP's automatic differentiation package vecvalder [8] ("`help MAPPautodiff`" in MAPP). By evaluating $\partial\vec{f}_e/\partial\vec{x}$, we can calculate conductances and plot the G/V curve shown in Fig. 8.

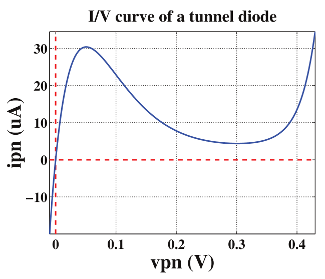


Fig. 7. I/V curve generated from the tunnel diode ModSpec model.

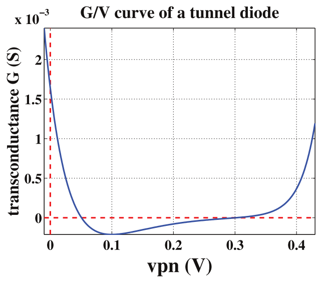


Fig. 8. G/V curve generated from the tunnel diode ModSpec model.

Furthermore, MAPP provides a Model Exerciser feature that allows users to plot curves like the ones in Fig. 7 and Fig. 8 with only a few lines of code. For example, we can initiate the model exerciser, then plot the I/V and G/V curves conveniently with the following MAPP code:

```
MEO = model_exerciser(Tunnel_Diode_ModSpec());
MEO.display(MEO); % displays available function names
          % and their usage, including 'ipn' and 'dipn_dvpn'
MEO.plot('ipn', 0:0.01:0.42, MEO);
MEO.plot('dipn_dvpn', 0:0.01:0.42, MEO);
```

Step 1.3: *Running the model within small circuits in MAPP*: Once the model has been examined standalone, it can be tested further in small circuits that are simulated in MAPP.

Frequently, this step reveals many problems (*e.g.*, bad numerics, unphysical results, connectivity issues, *etc.*) in newly-written models, especially those with equations that attempt to capture new physics. MAPP makes it easier to detect and address these problems than any other modelling/simulation framework we are aware of. MAPP's use of MATLAB® allows developers to debug their models effectively in an interactive coding environment. Also, MAPP's algorithm implementations are available to users as open source. They are object-oriented, mathematically-based, with functions inside clearly documented. They are meant to be easily accessible even by non-programmers. Thus debugging is more transparent to users. By running a new model within the DC, AC and transient algorithms within MAPP, most problems caused by the model for simulation can be detected.

For example, Fig. 9 shows a simple circuit where our previously introduced tunnel diode model is biassed within its negative resistance region by a voltage source, and connected with an RLC tank. With proper choice of parameters, the circuit becomes a negative-resistance LC oscillator. The circuit ("netlist") can itself be described in MAPP using MATLAB® commands ("`help MAPPcktnetlists`" for details). Running transient simulation in MAPP ("`help dot_transient`") demonstrates self-sustaining oscillation in the circuit, as illustrated in Fig. 10. From the standpoint of a device modeller, this provides important verification that the model can run in transient simulation. If transient fails, or if its results seem incorrect, the modeller gets to know immediately; the compact model's equations and/or their implementation can then be re-examined and corrected.

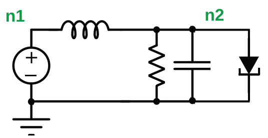


Fig. 9. Circuit schematic of an oscillator made with a tunnel diode.

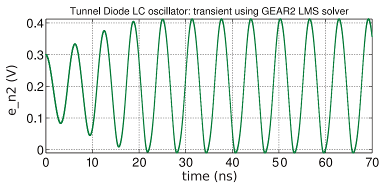


Fig. 10. Transient simulation results from the tunnel diode oscillator in Fig. 9.

Step 2: *Deploying the model*: Beyond providing facilities for testing models standalone and in small circuits, MAPP offers convenient and versatile tools to help prepare them for deployment. Most often, developers will wish to convert the ModSpec model into Verilog-A, the current industry standard for compact modelling [12], for public release. The fact that the model has already been tested and debugged on small circuits makes it far more likely that a Verilog-A version, if written properly, will work well in simulation.

The most likely cause for Verilog-A model problems at this step are discrepancies between the Verilog-A model and its ModSpec version, which are not hard to introduce, even for experienced compact modellers. To aid debugging, MAPP comes with a translator, CoMeT (Compact Model Translator), that can translate Verilog-A to ModSpec. Since ModSpec is an executable format, the translated ModSpec model can be conveniently compared against the original one, both by evaluating model functions and by running circuit simulations. If the two ModSpec models, one original, the other auto-translated from Verilog-A, are found to be consistent, considerable confidence is generated that the released Verilog-A model is correctly implemented. If not, debugging the auto-translated ModSpec model in the interactive environment of MAPP makes it easy, typically, to locate problems in the Verilog-A version.

During automatic translation, CoMeT's Verilog-A parser extracts a directed acyclic graph (DAG) [13] for the model's equations[1]. For example, Fig. 11 shows the DAG generated for the tunnel diode. By helping modellers visualize code execution and variable/parameter parameter dependencies in the model, such graphs can be useful not only for understanding and debugging the model, but also for optimizing the code.

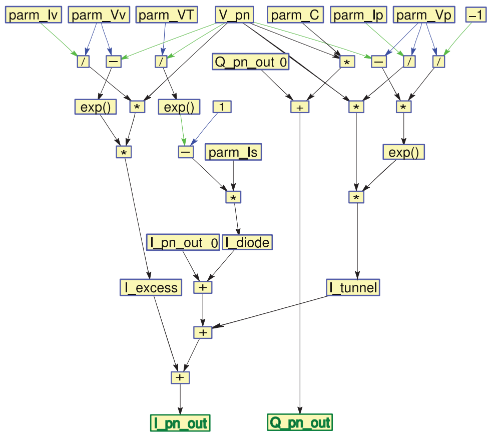


Fig. 11. DAG for the tunnel diode equations, generated by MAPP/CoMeT's Verilog-A translator/parser.

Thus, not only does MAPP incorporate Verilog-A within its compact modelling flow, it also adds convenient visualization, testing and debugging features that can speed development and improve the quality of Verilog-A versions of a compact model.

In parallel with Verilog-A release, model developers can also directly release their ModSpec models in MATLAB®. Moreover, a C++ version of ModSpec is also available, using which models can be compiled standalone to generate dynamically-loadable libraries that conform to a C++ version of the ModSpec API.

[1]MAPP's Verilog-A parser can also handle if-then-else statements and simple for loops.

Step 3: *Simulating the model*: Once the Verilog-A model is ready, it can be used by any simulator that supports compact model descriptions in Verilog-A. For example, the circuit in Fig. 9 was simulated in both Spectre and HSPICE. A snapshot of Spectre's results is shown in Fig. 12, while results from the HSPICE engine are plotted by MATLAB® in Fig. 13.

Furthermore, the C++ version of the ModSpec model can be simulated by any simulator that supports the C++ ModSpec API. We have implemented such support in the simulator Xyce [6] by writing a ModSpec interface within Xyce. This Xyce-ModSpec interface consists of less than 1000 lines of C++ code, and can dynamically link any C++ ModSpec model into Xyce. Fig. 13 overlays a Xyce simulation of the tunnel diode oscillator; a C++ ModSpec version of the tunnel diode model was incorporated into Xyce using the Xyce-ModSpec interface. We stress that results from the model, prototyped with MAPP, are identical in all simulators we have tried (Fig. 13 and Fig. 12), with Verilog-A and ModSpec deployments being consistent. The MAPP-based development flow we have outlined makes it far easier and faster to achieve such consistency than previous flows.

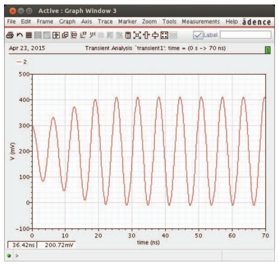


Fig. 12. Screenshot from Cadence® Virtuoso®, showing Spectre transient simulation results of the circuit in Fig. 9.

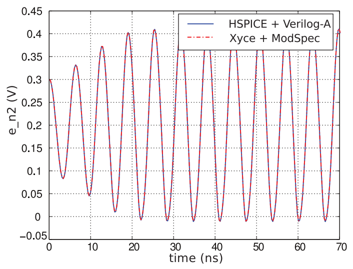


Fig. 13. Transient results from HSPICE and Xyce of the circuit in Fig. 9.

The ModSpec model format, being relatively new, is not widely supported in simulators yet. Nevertheless, its adoption can confer a number of advantages. Implementing a C++ ModSpec interface in a simulator is typically much easier than implementing Verilog-A support; once the former is done, any ModSpec model can be immediately used by the simulator by linking in its shared library dynamically. Supplying a model that conforms to the open and full-featured ModSpec API improves compatibility across different simulators; proprietary models can be deployed as pre-compiled dynamically-loadable binary libraries to help protect intellectual property (IP). Since ModSpec API functions can be called directly, without relying on any particular simulator or analysis, deployed models can tested standalone – another important features. These merits make ModSpec-based model deployment a useful complement to Verilog-A releases.

III. Prototyping Simulation Algorithms in MAPP

MAPP comes with features that also make prototyping simulation algorithms quick and convenient:

○ Simulation algorithms in MAPP rely only on the API of MAPP's DAE objects, as noted earlier; this API does not expose any details of device models or network formulations. This makes it possible, and easy, to write powerful algorithms that apply immediately to any system or device. Device models and Equation Engine code do not need to be modified, or even consulted, when algorithms are written or updated. Various *ad-hoc* concepts used in traditional circuit simulators, such as SPICE's Norton-Theorem-based RHS [14], companion equivalent circuits, *etc.*, are eliminated, making algorithms much more simple and elegant.

- All simulation algorithms in MAPP are object-oriented. The encapsulation induced by such object-oriented implementations enforces modularity, improves code readability and simplifies code documentation. It also enables inheritance between analyses, *i.e.*, developers of higher-level algorithms can easily leverage more basic ones using only a few lines of code. Fig. 14 depicts how MAPP's simulation algorithms are structured — starting from the basic numerical routines shown on the top of Fig. 14, many algorithms, from the standard DC, AC, transient analyses to more advanced ones, build on others hierarchically. This feature greatly reduces the time and trouble it takes to develop or prototype advanced new algorithms, allowing MAPP to easily support more of them than most other simulators.

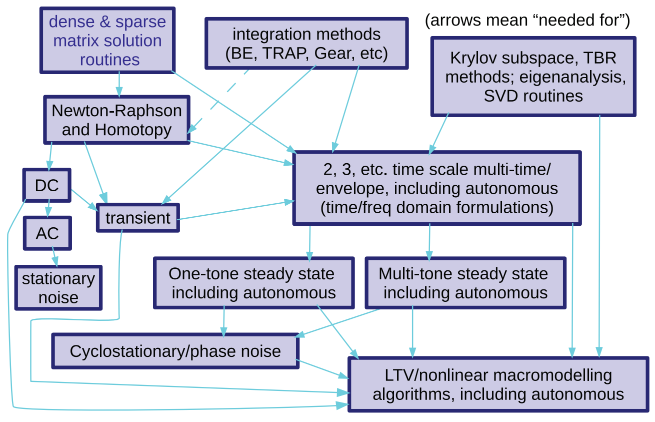

dense & sparse matrix solution routines
integration methods (BE, TRAP, Gear, etc)
(arrows mean "needed for")
Krylov subspace, TBR methods; eigenanalysis, SVD routines
Newton-Raphson and Homotopy
DC
AC
stationary noise
transient
2, 3, etc. time scale multi-time/ envelope, including autonomous (time/freq domain formulations)
One-tone steady state including autonomous
Multi-tone steady state including autonomous
Cyclostationary/phase noise
LTV/nonlinear macromodelling algorithms, including autonomous

Fig. 14. Structuring of MAPP's simulation algorithms.

In the remainder of this section, we illustrate these features of MAPP using the shooting algorithm as an example.

A. Simulation Algorithm Example: the Shooting Method

The shooting method (henceforth just "shooting") [15] is a numerical algorithm for finding Periodic Steady-State (PSS) responses of systems. Shooting poses the problem of finding a periodic response as that of finding an (initially unknown) initial condition that evolves to itself after one period. In other words, denoting the initial condition by $\vec{x}_0$, shooting can be written as

$$\vec{g}(\vec{x}_0) \triangleq \vec{x}(T) - \vec{x}_0 = \vec{0}, \tag{8}$$

where $\vec{x}(t)$ is the solution of the system DAE with initial condition $\vec{x}_0$, *i.e.*, $\vec{x}(t)$ satisfies

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t), \vec{u}(t)) = \vec{0}, \tag{9}$$

$$\text{and} \quad \vec{x}(0) = \vec{x}_0. \tag{10}$$

$\vec{g}(\vec{x}_0)$ is an algebraic function of $\vec{x}_0$; as such, it can be solved using numerical algorithms for nonlinear algebraic systems, such as the Newton-Raphson (NR) method [7]. Since evaluating $\vec{g}(\cdot)$ at each NR step involves running a transient simulation, shooting, in essence, reduces the PSS problem (a boundary value problem) to a few initial value problems.

Algorithm 1 Shooting Algorithm in MAPP (pseudo-code)

```
shootObj = shoot(DAE): // constructor
 1: shootObj.DAE = DAE;
 2: shootObj.tranObj = LMS(DAE); // transient simulation object
 3: set up member functions: .solve, .g, and .J
 4: return shootObj;

shootObj.solve (initguess, T):
 1: x0 ← NR(@g, @J, initguess);
 2: shootSols = tranObj.solve(x0, 0, T);
 3: return shootSols;

shootObj.g (x0):
 1: tranSols = tranObj.solve(x0, 0, T);
 2: return gout = tranSols(:, n) - x0;

shootObj.J (x0):
 1: tranSols = tranObj.solve(x0, 0, T);
 2: Ci_pre = DAE.dq_dx(x0);
 3: M = eye(n);
 4: for i = 2:n do
 5:     x = tranSols(:, i); u = inputs(:, i);
 6:     Ci = DAE.dq_dx(x); Gi = DAE.df_dx(x, u);
 7:     M = (Ci + (tpts(i) - tpts(i-1)) * Gi) \ Ci_pre * M;
 8:     Ci_pre = Ci;
 9: end for
10: return Jout = M - eye(n);
```

Algorithm 1, showing pseudo-code for MAPP's implementation of shooting, further highlights how algorithm prototyping is quick and convenient in MAPP:

- Shooting in MAPP is formulated using DAEs and is unrelated to device models or physical domains.
- Shooting requires running transient simulations. But since algorithm implementations in MAPP are object-oriented, transient analysis does not need to be re-implemented within shooting. Instead, a transient analysis object is initiated and its methods called. Likewise, shooting itself is also written in an object-oriented manner, with its numerical routines encapsulated in its member functions. So other analyses, if needed, can also internally use shooting conveniently.
- MAPP's implementation of shooting leverages MATLAB's vector (line 5 in shootObj.J) and sparse matrix (line 7 in shootObj.J) data types and associated functions. This makes the actual code almost as simple as the pseudo-code shown in Algorithm 1.

Shooting was implemented, debugged and tested on a number of circuits in about a week by one of the authors. By way of comparison, it had taken a bright, hard-working graduate student two years to implement shooting in SPICE3 [16]; the implementation was not widely released because the student was unable to debug it satisfactorily before he graduated [17]. In fact, more than twenty years later, shooting is still not widely available in open-source simulators today.

IV. Results

In this section, we provide more examples of device models and simulation algorithms prototyped in MAPP.

A. Device Example: MOSFET Models in MAPP

Fig. 15 shows the characteristic curves of several MOSFET models implemented in MAPP: BSIM6.1.0 [18], PSP level 103, version 3.0 [19], [20], the MIT Virtual Source (MVS) model, version 1.0.1 [21], MOS11 level 1101 version 2 [22]. The MVS model was hand-coded in ModSpec based on equations published in [21]; the rest were ported into MAPP using CoMeT, its Verilog-A parser and translator. The characteristic curves shown were produced by MAPP's model exerciser, using the default parameters of each model. All the models were also confirmed to work in various circuits and analyses.

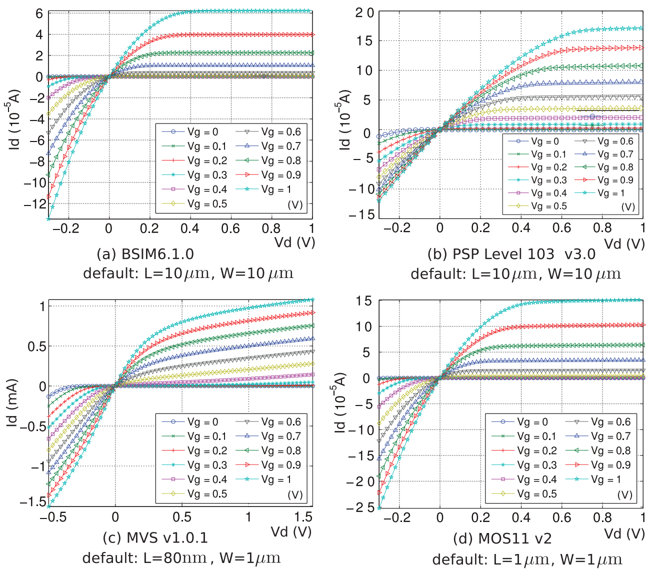


Fig. 15. Characteristic curves of several MOSFET models in MAPP.

B. Algorithm Example: Homotopy/Continuation Analysis

In this example, we use MAPP to analyze a Goto pair, [23], *i.e.*, a series connection of two tunnel diodes (Fig. 16). The negative resistance regions of the diodes make the circuit feature two stable operating points (along with one unstable one) — *i.e.*, the voltage at node `n2` can settle to one of two possible values, as illustrated in Fig. 16. While traditional DC operating point analysis can only find one solution at a time given a specific initial guess, a class of techniques named homotopy or continuation, adapted for circuit simulation [24], is capable of finding multiple DC operating points by "tracking" the curves formed by solutions in state space.

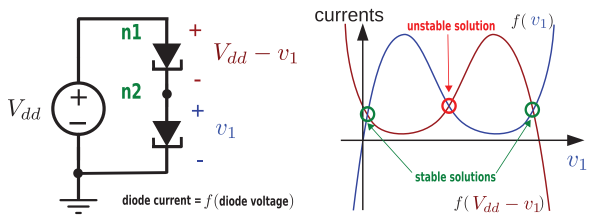


Fig. 16. A Goto pair circuit features multiple DC solutions.

We have implemented homotopy/continuation analysis in MAPP. For the case of identical diodes, Fig. 17 shows how homotopy finds multiple operating points, as Vdd (labelled `Vdd:::E` in the figure) is swept from 0 to 1V.

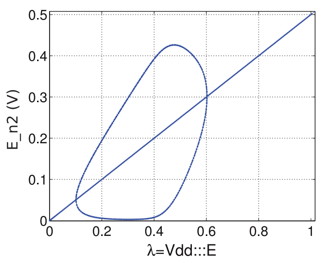


Fig. 17. All solutions of E(n2) (also denoted v_1 in Fig. 16) at different Vdd values, generated by arclength continuation algorithm in MAPP. For Vdd $\in [0.1, 0.6]$, there are three solutions, as predicted by Fig. 16.

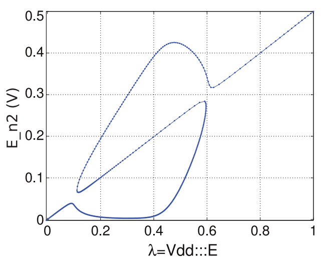


Fig. 18. All solutions of E(n2) in the Goto pair when the two diodes have slightly different parameter values.

Note that in Fig. 17, starting from the origin, the solution curve splits into three branches when Vdd is around 0.1V, and the branches converge when Vdd is around 0.6V. As is discussed in [24], this trifurcation is a probability-0 event that disappears if the two diodes are not identical — this situation is shown in Fig. 18.

C. Multi-physics Example: Optical Ring Resonator

MAPP is not limited to electrical circuits. As an example, here we consider an optical system — a silicon ring resonator [25] shown in Fig. 19.

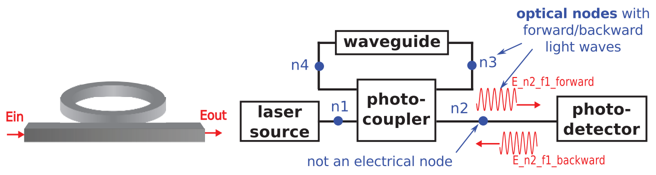


Fig. 19. Silicon micro-ring resonator and its system diagram used in MAPP. The coupling region is modelled as a photo-coupler.

This optical system is modelled in MAPP as a network of component devices. However, the connections in the network are not electrical nodes; they are not associated with voltages or currents. Instead, each optical connection in Fig. 19 represents a set of travelling waves of the light's electric field. MAPP models these waves as complex numbers, or phasors, whose modulus and angle represent the magnitude and phase shift of light. Each wave can be comprised of several phasors, at different frequencies, each potentially varying with time (*i.e.*, representing time-varying envelopes). Each optical node is associated with one forward[2] and one backward wave. For example, in Fig. 19, `E_n2_f1_forward` represents the light component at frequency `f1` that is travelling from the photocoupler to the photodetector.

Rather than voltages and currents, it is such forward and backward waves that constitute the I/Os of optical ModSpec devices (*i.e.*, in (5) and (6)). MAPP's Optical Equation Engine (which is separate from its Electrical Equation Engines) sets up system equations in terms of such waves for connected networks of optical elements. Since these system equations are simply DAEs, all simulation algorithms in MAPP continue to work. As an example, Fig. 20 plots the relative magnitude of `E_n2_f1_forward` vs the laser source's frequency `f1` as it is swept over a range. The peaks and nulls result from constructive and destructive interference between light waves propagating through the system.

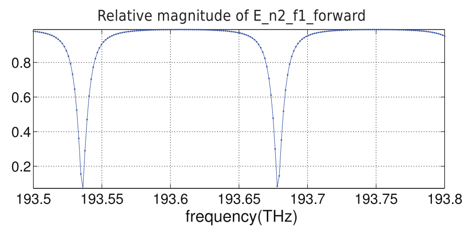


Fig. 20. Frequency sweep on the ring resonator in Fig. 19. The relative magnitude of the electric field of the light arriving at the photodetector with respect to that emitted by the source is plotted.

V. Summary

In this paper, we have presented Berkeley MAPP, a MATLAB®-based platform for prototyping device models and simulation algorithms. We have described a MAPP-based device modelling flow that eases and speeds up the process of developing high-quality, simulation-ready models. Prototyping analyses is also far more convenient in MAPP than in Berkeley SPICE and similar simulators. We have illustrated some of MAPP's algorithmic capabilities and usability features. MAPP is available as open source; it is hoped that it will facilitate research and teaching in the fields of compact modelling and simulation.

Acknowledgments

We thank Colin McAndrew for many useful comments on an initial draft of the paper. This work was supported through the NCN-NEEDS program, which is funded by the NSF (contract 1227020-EEC) and by the SRC.

[2]Directionality is determined based on system topology by the optical equation engine.

References

[1] L.W. Nagel. *SPICE2: a computer program to simulate semiconductor circuits*. PhD thesis, EECS department, University of California, Berkeley, Electronics Research Laboratory, 1975. Memorandum no. ERL-M520.

[2] D. O. Pederson and A. Sangiovanni-Vincentelli. SPICE 3 Version 3F5 User's Manual. *Dept. EECS, Univ. California, Berkeley, CA*, 1994.

[3] P. Nenzi and H. Vogt. Ngspice Users Manual Version 26 (Describes ngspice-26 release version). 2014.

[4] A. Davis. The gnu circuit analysis package. 2006. http://www.gnu.org/software/gnucap.

[5] M. E. Brinson and S. Jahn. Qucs: A GPL software package for circuit simulation, compact device modelling and circuit macromodelling from DC to RF and beyond. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 22(4):297–319, 2009.

[6] E. R. Keiter, T. Mei, T. V. Russo, R. L. Schiek, H. K. Thornquist, J. C. Verley, D. A. Fixel, T. S. Coffey, R. P. Pawlowski, C. E. Warrender, et al. Xyce parallel electronic simulator users' guide, Version 6.0. 1. Technical report, Raytheon, Albuquerque, NM; Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2014.

[7] Jaijeet Roychowdhury. Numerical simulation and modelling of electronic and biochemical systems. *Foundations and Trends in Electronic Design Automation*, 3(2-3):97–303, December 2009.

[8] D. Amsallem and J. Roychowdhury. ModSpec: An open, flexible specification framework for multi-domain device modelling. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 367–374. IEEE, 2011.

[9] Bichen Wu and J. Roychowdhury. Efficient per-element distortion contribution analysis via harmonic balance adjoints. In *Proc. IEEE CICC*, pages 1–4, Sept 2014.

[10] MAPP: The Berkeley Model and Algorithm Prototyping Platform. http://mapp.eecs.berkeley.edu.

[11] S. M. Sze and K. K. Ng. *Physics of semiconductor devices*. John Wiley and Sons, 2006.

[12] L. Lemaitre, G. Coram, C. C. McAndrew, and K. Kundert. Extensions to Verilog-A to support compact device modeling. In *Behavioral Modeling and Simulation, 2003. BMAS 2003. Proceedings of the 2003 International Workshop on*, pages 134–138. IEEE, 2003.

[13] S. Touati and B. de Dinechin. *Advanced Backend Optimization*. ISTE. John Wiley and Sons, 2014.

[14] A. Vladimirescu. *The SPICE book*. John Wiley & Sons, Inc., 1994.

[15] S. Skelboe. Computation of the periodic steady-state response of nonlinear networks by extrapolation methods. *IEEE Trans. Ckts. Syst.*, CAS-27:161–175, 1980.

[16] Pranav N Ashar. Implementation of algorithms for the periodic-steady-state analysis of nonlinear circuits. Technical report, 1989.

[17] P. Kinget. Private communication, June 1997.

[18] H. Agarwal, S. Khandelwal, J. P. Duarte, Y. S. Chauhan, A. Niknejad, and C. Hu. BSIM6.1.0 MOSFET Compact Model. 2014. http://www-device.eecs.berkeley.edu/bsim/?page=BSIM6_LR.

[19] PSP release: Level 103 v3.0, 2014. http://psp.ewi.tudelft.nl.

[20] Gennady Gildenblat, Weimin Wu, Xin Li, Ronald van Langevelde, Andries J Scholten, Geert DJ Smit, and Dirk BM Klaassen. Surface-potential-based compact model of bulk mosfet. In *Compact Modeling*, pages 3–40. Springer, 2010.

[21] S. Rakheja and D. Antoniadis. MVS Nanotransistor Model (Silicon), Oct 2014.

[22] G. Coram. MOS Model 11 Level 1101. http://www.designers-guide.com/VerilogAMS.

[23] Roy Hakim, Elad D. Mentovich, and Shachar Richter. Towards Post-CMOS Molecular Logic Devices. In Nicolas Lorente and Christian Joachim, editors, *Architecture and Design of Molecule Logic Gates and Atom Circuits*, Advances in Atom and Single Molecule Machines, pages 13–24. Springer Verlag, 2013.

[24] J. Roychowdhury and R. Melville. Delivering Global DC Convergence for Large Mixed-Signal Circuits via Homotopy/Continuation Methods. *IEEE Trans. on Computer-Aided Design*, 25:66–78, Jan 2006.

[25] E. Kononov. *Modeling photonic links in Verilog-A*. PhD thesis, Massachusetts Institute of Technology, 2013.