Verifying Simulation Models with Optimization

David Long

With advice and comments from:
Peter Feldmann
Bob Melville
Jaijeet Roychowdhury

April, 2002

One minute summary

Models for circuit-level simulators are complicated.

Simulators don’t respect reality, so the models have to work in regions that don’t make physical sense.

The models need to have nice numeric properties in order to help the simulator out as much as possible.

Verifying the models is a pain.

Optimization can help.

What is a device model?

It gives the behavior of something like a transistor in a circuit. The model represents relationships among voltages, currents, charges, fluxes, etc.

Computationally, it's a box of equations, dependent on:

- ⋆ process parameters (doping density, oxide thickness, . . .);
- ⋆ geometric parameters (device size, . . .);
- ⋆ environment parameters (temperature, . . .).

A simulator calls the model with, e.g., the voltage V at the device terminals.

The model computes the current $I + d\,Q/d\,t$ that would flow into the device from the terminals.

Why are device models complicated?

They have to work for lots of different processes, devices geometries and sizes, operating regimes, etc.

They need to handle lots of “non-ideal” effects, especially in more advanced technologies.

BSIM3, a modern MOS transistor model, has about 140 inputs (parameters and terminal voltages) and 500 “essential” lines of code (equations).

What do simulators want?

Before the simulation has converged, models can be exercised in unreasonable ways.

- ⋆ Models must be robust (always evaluatable).
- ⋆ If they behave in physically reasonable ways, that's a bonus.

Simulators like smoothness:

- ⋆ It enhances convergence.
- ⋆ It is required for some numerical methods.

Users like it too, since nature generally behaves in a smooth manner.

What do we want to verify?

We don’t much care whether the model matches measurements...

Basic physical properties of the device:

- ⋆ passivity $\sum_j I_j V_j \geq 0$
- ⋆ symmetry $I_{ds}(v_d, v_s) = -I_{ds}(v_s, v_d)$
- ⋆ monotonicity $d\, I_{ds} / d\, v_g \geq 0$

Evaluatability:

- ⋆ no division by zero
- ⋆ arguments to special functions must be in range

Numerical niceness:

- ⋆ no overflows
- ⋆ no catastrophic cancellation
- ⋆ expressions and derivatives should be continuous

How can optimization help?

Desirable properties can be expressed in terms of optimization problems.

The optimization is *targeted* at things that can cause problems.

- ⋆ Some checks focus on subexpressions, not just the final output.

In contrast, random playbacks do not try to cover worst-case behavior.

How do we check properties with optimization?

Check $\sum_j I_j V_j \geq 0$ by verifying min $\sum_j I_j V_j \geq 0$.

$I_{ds}(v_d, v_s) = -I_{ds}(v_s, v_d)$ when max $\|I_{ds}(v_d, v_s) + I_{ds}(v_s, v_d)\| = 0$.

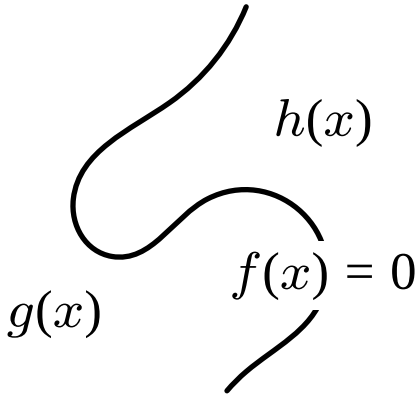
log $f(x)$ is always evaluatable when min $f(x) > 0$.

$f(x) - g(x)$ is ill-conditioned when $\|f(x) - g(x)\| < \epsilon\,(\|f(x)\| + \|g(x)\|)$.

What about continuity?

That's the most difficult task.

Assuming that all the primitives are continuous, the problem arises from conditional expressions: $(f(x) > 0)\ ?\ g(x) : h(x)$.



Maximize $||g(x) - h(x)||$ subject to the constraint $f(x) = 0$.

But $g(x)$ may not even be evaluatable when $f(x) \leq 0 \ldots$

How do we handle partial functions?

In general, I have no idea.

In practice, a lot of the cases are simple.

Whenever you have $(f(x) > 0)\ ?\ g(x)\ :\ h(x)$, replace subexpressions involving $f(x)$ in $g(x)$ and $h(x)$.

- $\star$ In $g(x)$, use max($f(x), \epsilon$).
- $\star$ In $h(x)$, use min($f(x), -\epsilon$).

The same technique is needed for evaluatability checks in $g(x)$ and $h(x)$.

What are the numeric issues?

Possibly there are a large number of inputs (100s).

Some models have internal unknowns.

Some devices have very strong nonlinearities.

Sometimes the bad parts of the space are “small.”

When do we do the verification?

Ideally, we'd like to verify the model up front as much as possible, before deploying it in a simulator.

When we do this, the number of inputs is large.

In contrast, after parameter extraction, there are only a small number of inputs (generally temperature and terminal voltages).

Some physical properties may not hold for all parameter settings, and it can be difficult to describe exactly when they do hold.

Evaluatability and numerical niceness tests can generally be done up front though.

How do we handle internal unknowns?

When checking physical properties, it may be necessary that internal unknowns have physically reasonable values before the properties hold.

In such cases, each step of the inner loop of the optimization must include a solve for the internal unknowns.

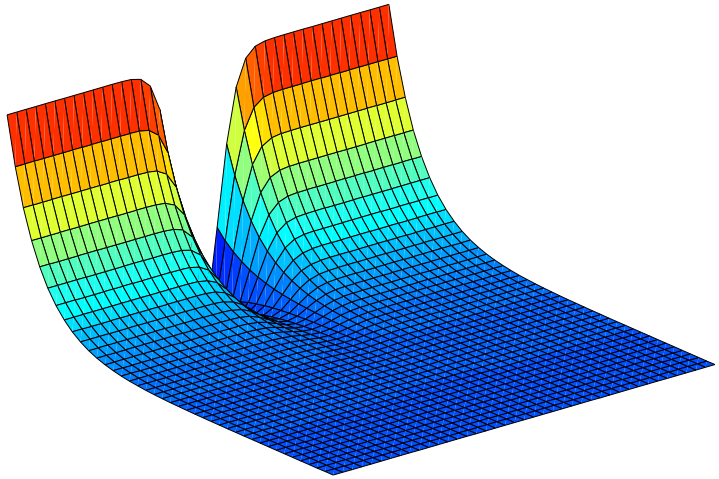
To get the reduced Jacobian (involving only terminals), linearize and use a standard sensitivity analysis.

During evaluatability and numerical niceness tests, we treat the internal values as externally visible.

When do the optimizers have problems?

One issue with properties like passivity is that devices generally have an “off” state that is strongly attractive. It’s easy to get a local minimum for power by turning the device off.

This problem is exacerbated because of the strong nonlinearities involved.



Unless you get lucky, you wind up on the flat edge.

How do we avoid these problem?

I have no idea. I wound up just taking lots of random starting points and hoping.

I tried interval methods, but there's enormous reconvergent fanout in the expressions, so the intervals are very conservative. I tried various strategies to get tighter bounds.

- ⋆ Compute monotonicity information.
- ⋆ Scan the expression DAG for macro functions (local reconvergence).
- ⋆ Use combinations of interval representations.

In the end, none of this was sufficient.

A plea to model makers

Write equations, not code.

It's easy to generate code from equations, and the two are mostly guaranteed to agree.

It's a @#$! to generate equations from code, and the two never agree.

Equations make life easy for many people.

- ⋆ People who write simulators can add your model without errors.
- ⋆ Users can understand the equations better.
- ⋆ Verification like I've described requires equations.