An Efficient Method to Simulate Threshold-Crossing Events

G. Peter Fang
Texas Instruments, Inc.
g-fang1@ti.com

ABSTRACT

In this work, a new and efficient method is developed to simulate threshold-crossing events (for example, the cross event in Verilog-A/MS). The method converges directly to the solution at the crossing without the need of computing additional timepoints. The violated cross condition is solved together with the circuit equations as one system of nonlinear equations and the timestep is treated as an independent variable. This method has very little overhead and can be easily integrated into an existing simulation flow. Simulation results show good convergence rate - less than two extra Newton iterations needed to accurately locate the crossing. This method could speedup transient simulation significantly when there are a large number of threshold-crossing events.

1. INTRODUCTION

The cross event has been widely used in behavior-level modeling and simulation. It changes the dynamics of the model when certain analog expression crosses through zero. The cross function in Verilog-A generates a monitored analog event to detect threshold crossing. It controls the timestep to accurately resolve the crossing. The cross function is also used in the A2D event in Verilog-AMS for threshold crossing [1].

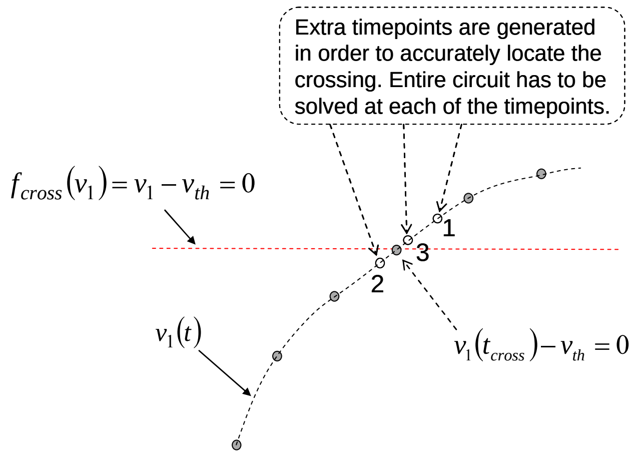
Extra timepoints are generated in order to accurately locate the crossing. Entire circuit has to be solved at each of the timepoints.
$f_{cross}(v_1) = v_1 - v_{th} = 0$
1
2
3
$v_1(t)$
$v_1(t_{cross}) - v_{th} = 0$

Figure 1. The existing method needs to compute additional timepoints to accurate resolve the crossing

The best known method for locating the crossing points works as follows: The simulator checks all cross conditions at each timepoint. If a violation is detected, the simulator rejects the timepoint, enters a search mode and tries to hit the timepoint when the crossing is occurring. As illustrated in Figure 1, the simulator iteratively predicts (via interpolation) and solves the crossing point until the errors are within tolerances. It normally takes 2-4 timepoints (sometimes much more) before an accurate crossing can be found. The entire circuit or system has to be evaluated and solved at those timepoints. A large number of cross events could slow down a transient simulation significantly [2].

In this work, we developed a more rigorous method to solve the crossing problem. Instead of computing solutions at a number of timepoints and checking the cross condition at each timepoint, we solve the cross condition together with the circuit equations and treat the timestep as an independent variable. The method directly converges to the solution at the crossing, thus is more efficient than the existing methods.

2. COUPLED NEWTON METHOD

Most circuit simulators use modified nodal analysis (based on Kirchhoff's laws) to formulate a system of N differential algebraic equations (DAEs) [3],

$$\bar{f}_{ckt}(\bar{v}(t)) = \frac{d}{dt}\bar{q}(\bar{v}(t)) + \bar{i}(\bar{v}(t)) + \bar{u}(t) = 0\,, \qquad (1)$$

where $\bar{u} \in R^N$ is the vector of input sources, $\bar{v} \in R^N$ is the vector of solution variables, and $\bar{i}, \bar{q} \in R^N$ are the vectors of resistive currents and node charges/branch fluxes. The time derivative term can be discretized using a time integration scheme. Without loss of generality, we only consider backward Euler scheme for simplicity. The resulting system of nonlinear equations is

$$\bar{f}_{ckt}(\bar{v}_m) = \frac{\bar{q}(\bar{v}_m) - \bar{q}(\bar{v}_{m-1})}{h_m} + \bar{i}(\bar{v}_m) + \bar{u}_m = 0\,, \qquad (2)$$

where m is the time index. And the cross condition is an explicit equation of solution variables, i.e., node voltages and branch currents, given by,

$$f_{cross}(\bar{v}_m) = 0\,. \qquad (3)$$

In order to locate the crossing and compute the solution at the crossing simultaneously, we treat the timestep h_m as an independent variable or an unknown, and solve the cross condition together with circuit equations as one system of nonlinear equations,

$$\begin{cases} \bar{f}_{ckt}(\bar{v}_m, h_m) = 0 \\ f_{cross}(\bar{v}_m) = 0 \end{cases} \tag{4}$$

or

$$\bar{F}_{coupled}(\bar{v}_m, h_m) = 0 \tag{5}$$

where $\bar{F}_{coupled} \in R^{N+1}$, i.e., we now solve N+1 nonlinear equations for N+1 unknowns.

Applying Newton method to the above equation, we obtain,

$$\begin{pmatrix} \dfrac{\partial \bar{f}_{ckt}}{\partial \bar{v}_m} & \dfrac{\partial \bar{f}_{ckt}}{\partial h_m} \\ \dfrac{\partial f_{cross}}{\partial \bar{v}_m} & 0 \end{pmatrix} \begin{pmatrix} \Delta \bar{v}_m^{k+1} \\ \Delta h_m^{k+1} \end{pmatrix} = -\begin{pmatrix} \bar{f}_{ckt}(\bar{v}_m^k, h_m^k) \\ f_{cross}(\bar{v}_m^k) \end{pmatrix} \tag{6}$$

where $\dfrac{\partial \bar{f}_{ckt}}{\partial \bar{v}_m} \in R^{N\times N}$, $\dfrac{\partial \bar{f}_{ckt}}{\partial h_m} \in R^{N\times 1}$, and $\dfrac{\partial f_{cross}}{\partial \bar{v}_m} \in R^{1\times N}$. And $\overline{\overline{J}}_{ckt} = \dfrac{\partial \bar{f}_{ckt}}{\partial \bar{v}_m}$ is the original circuit Jacobian matrix.

Multiplying both sides of the above equation with

$$\begin{pmatrix} \overline{\overline{J}}_{ckt}^{-1} & 0 \\ 0 & 1 \end{pmatrix},$$

we obtain

$$\begin{pmatrix} \overline{\overline{I}}_{N\times N} & \overline{\overline{J}}_{ckt}^{-1} \dfrac{\partial \bar{f}_{ckt}}{\partial h_m} \\ \dfrac{\partial f_{cross}}{\partial \bar{v}_m} & 0 \end{pmatrix} \begin{pmatrix} \Delta \bar{v}_m^{k+1} \\ \Delta h_m^{k+1} \end{pmatrix} = -\begin{pmatrix} \overline{\overline{J}}_{ckt}^{-1} \bar{f}_{ckt}(\bar{v}_m^k, h_m^k) \\ f_{cross}(\bar{v}_m^k) \end{pmatrix}. \tag{7}$$

Performing Gauss elimination, the equation becomes,

$$\begin{pmatrix} \overline{\overline{I}}_{N\times N} & \overline{\overline{J}}_{ckt}^{-1} \dfrac{\partial \bar{f}_{ckt}}{\partial h_m} \\ 0 & -\dfrac{\partial f_{cross}}{\partial \bar{v}_m}\left(\overline{\overline{J}}_{ckt}^{-1} \dfrac{\partial \bar{f}_{ckt}}{\partial h_m}\right) \end{pmatrix} \begin{pmatrix} \Delta \bar{v}_m^{k+1} \\ \Delta h_m^{k+1} \end{pmatrix} = -\begin{pmatrix} \overline{\overline{J}}_{ckt}^{-1} \bar{f}_{ckt} \\ f_{cross} - \dfrac{\partial f_{cross}}{\partial \bar{v}_m}\left(\overline{\overline{J}}_{ckt}^{-1} \bar{f}_{ckt}\right) \end{pmatrix} \tag{8}$$

We can perform backward substitution to obtain solution,

$$\Delta h_m^{k+1} = \frac{f_{cross} + \dfrac{\partial f_{cross}}{\partial \bar{v}_m}\left(\overline{\overline{J}}_{ckt}^{-1}\left(-\bar{f}_{ckt}\right)\right)}{\dfrac{\partial f_{cross}}{\partial \bar{v}_m}\left(\overline{\overline{J}}_{ckt}^{-1} \dfrac{\partial \bar{f}_{ckt}}{\partial h_m}\right)}, \tag{9}$$

and

$$\Delta v_m^{k+1} = \overline{\overline{J}}_{ckt}^{-1}\left(-\bar{f}_{ckt}\right) + \left(\overline{\overline{J}}_{ckt}^{-1} \frac{\partial \bar{f}_{ckt}}{\partial h_m}\right)\Delta h_m^{k+1}. \tag{10}$$

Note that the first term $\overline{\overline{J}}_{ckt}^{-1}\left(-\bar{f}_{ckt}\right)$ is the solution of the circuit equations when the timestep is not treated as an unknown. And the second term is the correction to the solution due to the change of the timestep. Most modern circuit simulators perform in-place LU factorization. So the term $\overline{\overline{J}}_{ckt}^{-1} \dfrac{\partial \bar{f}_{ckt}}{\partial h_m}$ can be easily obtained by standard forward and backward substitutions of the circuit Jacobian matrix. The overhead for solving the cross condition are those substitutions, whose computational cost is really negligible compared with the cost of a full LU factorization. We would like to point out that this method works with any linear solvers in a given circuit simulator, although the computational cost may vary.

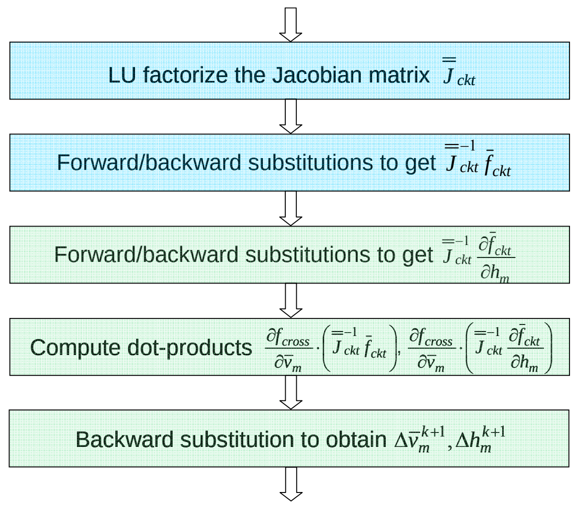
LU factorize the Jacobian matrix $\overline{\overline{J}}_{ckt}$
Forward/backward substitutions to get $\overline{\overline{J}}_{ckt}^{-1} \bar{f}_{ckt}$
Forward/backward substitutions to get $\overline{\overline{J}}_{ckt}^{-1} \dfrac{\partial \bar{f}_{ckt}}{\partial h_m}$
Compute dot-products $\dfrac{\partial f_{cross}}{\partial \bar{v}_m}\cdot\left(\overline{\overline{J}}_{ckt}^{-1} \bar{f}_{ckt}\right), \dfrac{\partial f_{cross}}{\partial \bar{v}_m}\cdot\left(\overline{\overline{J}}_{ckt}^{-1} \dfrac{\partial \bar{f}_{ckt}}{\partial h_m}\right)$
Backward substitution to obtain $\Delta \bar{v}_m^{k+1}, \Delta h_m^{k+1}$

Figure 2. Procedure to solve the coupled linear system

As shown in Figure 2, only a few additional steps (one forward substitution and one backward substitution, and two dot products) are needed to solve the coupled nonlinear system. This method can be easily integrated into an existing simulation flow with a small amount of effort.

3. IMPLEMENTATION DETAILS

3.1 Crossing Detection

The common crossing detection technique checks signs of a cross condition before and after a timestep. A sign change indicates that a crossing has occurred and a search algorithm is activated to precisely locate the crossing. This method assumes cross conditions vary linearly between two timepoints. Yet for most cases those cross conditions are polynomials. If a crossing occurs within a timestep and the values at the beginning and the end of the step have the same sign, as illustrated in Figure 3, the simulator will step over the crossing without detecting it.

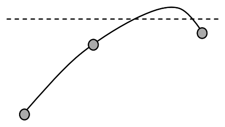

Figure 3. A missed crossing

Our implementation utilizes an interpolated second-order polynomial to find local maximum/minimum of the cross condition within a timestep, as shown in Figure 4. A crossing then can be detected by checking signs of maximum/minimum and the signal value at the beginning of the timestep. This method is one order of magnitude more accurate than previous method.

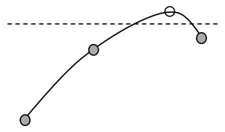

Figure 4. Detecting the crossing using local maximum

3.2 Modified Newton Flow

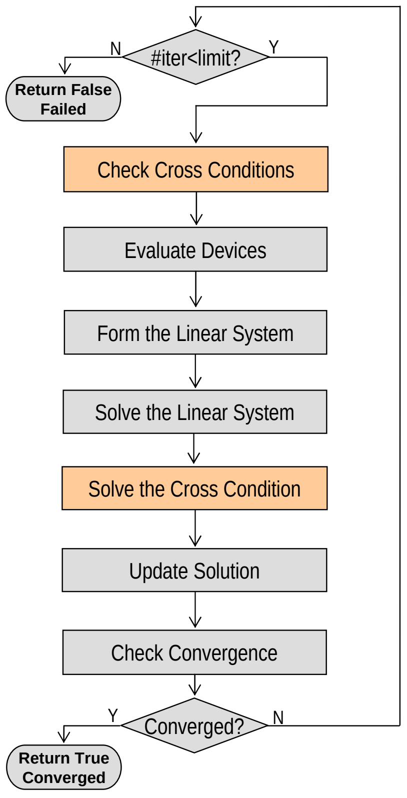

Figure 5. Modified Newton flow

The new method handles cross events inside the new loop and is transparent to the transient flow. A simplified Newton flow for the new method is shown in Figure 5. Note that two new blocks are inserted into the Newton loop to check and solve cross conditions.

As illustrated in Figure 6, the simulator evaluates and checks all cross conditions at the beginning of a Newton iteration. If a violation is detected, the simulator enters the search mode, projects a new timestep to hit the crossing (or the earliest crossing in the case of multiple violations) via interpolation. In search mode, the simulator also re-extrapolates using the new timestep, updates time-varying sources and time coefficients.

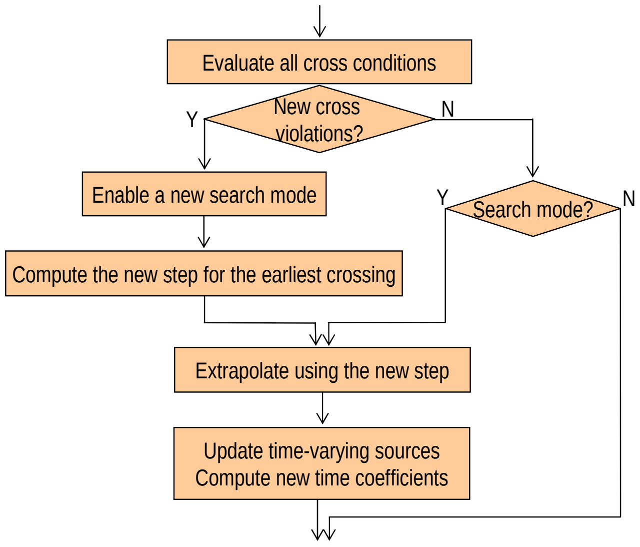
Evaluate all cross conditions
New cross violations?
Y
N
Enable a new search mode
Search mode?
Y
N
Compute the new step for the earliest crossing
Extrapolate using the new step
Update time-varying sources
Compute new time coefficients

Figure 6. Flow for checking cross conditions

In search mode, a coupled linear system is solved at each Newton iteration. The flow is shown in Figure 7. The simulator first computes derivatives of circuit equation with respect to the timestep. The cross condition part is then solved using equations (9) and (10). The timestep is updated before exit.

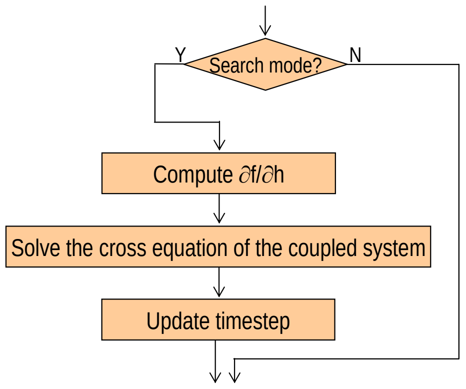

Figure 7. Flow for solving the cross condition

3.3 @cross in Verilog-A

The cross function in Verilog-A/MS [1] requires a timepoint to be placed just after the crossing, within tolerances, as shown in Figure 8. The coupled Newton method needs to be slightly modified to meet this requirement.

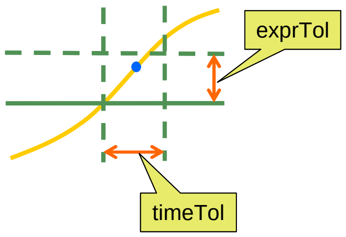

Figure 8. Relationship between time tolerance and expression tolerance in Verilog-A/MS

4. SIMULATION RESULTS

A basic version of the new method was implemented in our in-house circuit simulator, TISpice, and tested on a seven-stage ring oscillator shown in Figure 9.

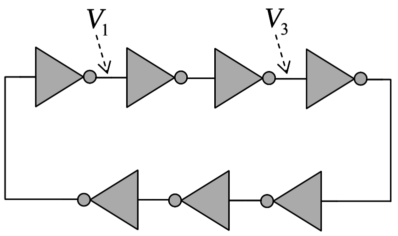

Figure 9. Seven-stage ring oscillator

We tested three cross conditions: $V_1 = 2$, $V_1 = V_3$, and $\frac{V_1^2}{1k\Omega} = 4mW$. The tolerances are expr_tol=1e-6 and time_tol = 1e-14. The second-order Gear method is used for time integration. The zero-crossing for the condition $f_{cross} = V_1 - V_3$ is shown in Figure 10.

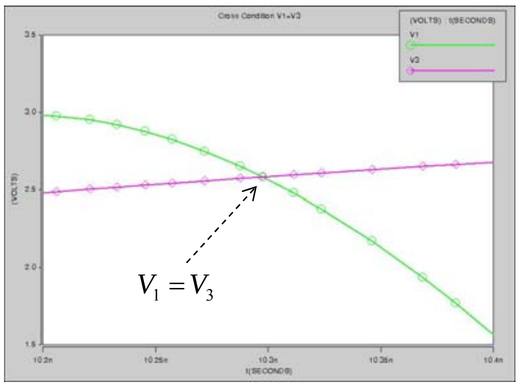

Figure 10. The zero-crossing for the condition $f_{cross} = V_1 - V_3$

The new method exhibits very good convergence rate for all three cross conditions – less than two extra Newton iterations are need to locate the crossing, while the best existing method normally requires 2-4 extra timepoints with 2-4 Newton iterations per timepoint. The average numbers of Newton iterations needed for each cross condition are summarized in Table 1.

Table 1. Average numbers of Newton iterations

cross condition	average # of Newton iterations at the crossing	average # of extra Newton iterations at the crossing
$V_1 = 2$	3.2	1.1
$V_1 = V_3$	3.5	1.4
$\frac{V_1^2}{1k\Omega} = 4mW$	3.4	1.3

5. CONCLUSION AND FUTURE WORKS

We developed a new, coupled Newton based method to simulate cross events. This method converges directly to the solution at the crossing and is theoretically superior to all the known methods. Initial results show very good convergence rate - less than two extra Newton iterations needed to accurately locate the crossing. The integration to an existing simulation flow is straightforward.

While we limit our discussion to threshold-crossing events in this paper, the new method can be used to solve general nonlinear DAE problems (nonlinear circuits or systems) with constraints that are explicit functions of solution variables, for example, accurate local maximum/minimum detection of a specified analog signal.

REFERENCES

[1] Verilog-AMS Language Reference Manual, version 2.2, November 2004.

[2] Ron Vogelsong, “Are your AMS behavioral modeling challenges surmountable?”, cdnusers.org, January 2005.

[3] William J. McCalla, Fundamentals of computer-aided circuit simulation, Kluwer, 1987.

APPENDIX: TIME DERIVITIVES

Backward Euler: $\frac{\partial \bar{f}_{ckt}}{\partial h_m} = \frac{-1}{h_m^2}\left(q(\bar{v}_m) - q(\bar{v}_{m-1})\right)$

Second-order Gear:

$$\frac{\partial \bar{f}_{ckt}}{\partial h_m} = \frac{-1}{h_m^2}\left(q(\bar{v}_m) - q(\bar{v}_{m-1})\right) - \frac{1}{(h_m + h_{m-1})^2}\left(q(\bar{v}_m) - q(\bar{v}_{m-2})\right)$$

Trapezoidal:

$$\frac{\partial \bar{f}_{ckt}}{\partial h_m} = \frac{-2}{h_m^2}\left(q(\bar{v}_m) - q(\bar{v}_{m-1})\right)$$