CONCERT: A Concurrent Transient Fault Simulator for Nonlinear Analog Circuits*

Junwei Hou and Abhijit Chatterjee
School of Electrical and Compute Engineering, Georgia Institute of Technology, Atlanta, GA
{jhou, chat}@ee.gatech.edu

Abstract

This paper presents a novel concurrent fault simulator (called CONCERT) for nonlinear analog circuits. Three primary techniques in CONCERT, including fault ordering, state prediction, and reduced-order fault matrix computation, greatly simplify fault simulation by making use of the residual similarities between the faulty and fault-free circuits. Between successive time steps, all circuits in the fault list are simulated concurrently before the simulator proceeds to the next time step. CONCERT also generates accurate fault coverage statistics that are tied to the circuit specifications. Up to two orders of magnitudes speedup are obtained for complete fault simulation, without any loss of accuracy. More speedup is achieved by CONCERT for evaluating the fault coverage of a test, using fault ordering and fault dropping technique.

1 Introduction

The fault simulation problem for nonlinear analog circuits is largely unsolved due to the complexity of analog simulation and the difficulties of simulating many analog faults simultaneously. In the digital world, concurrent fault simulation methods are well entrenched as the effects of multiple digital single-stuck-at faults can be propagated simultaneously from one gate to the next using only local information around the circuit nodes to which the fault effects have propagated. In contrast, analog faults typically affect voltage and current values across *all* the circuit nodes and branches, respectively, thereby making concurrent analog fault simulation very difficult. Currently, serial simulation of analog faults is the prevalent analog fault simulation methodology used in industry. As a consequence, comprehensive fault simulation of large mixed-signal circuits is almost impossible with today's tools.

In the digital domain, fault simulation algorithms are based on parallel fault simulation [12] and concurrent fault simulation [14] methods. In the analog domain, no fast fault simulation techniques have been reported for nonlinear circuits under general transient stimulus. Frequency domain (AC small signal) fault simulation of analog circuits linearized around the DC bias points for parameter tolerances is discussed in [13]. This approach is based on the assumption that the DC bias points of the faulty and fault-free circuits are the same, hence, disallowing large circuit parameter deviations under fault. Householder's formula [5] is used to assess the impact of the component tolerances on the AC response of the circuit under test (CUT). In [17], the authors have discussed the idea of concurrency which avoids re-evaluation of those components that have the same internal node values as in the fault-free case. A fault simulator called DRAFTS [9] has been developed for serial fault simulation of linear analog circuits. FLYER [15] reports significant improvement upon DRAFTS for fast fault simulation of linear analog circuits. A fast fault simulation method using fault ordering and circuit partitioning is reported in [16]. Householder's formula has also been applied to analyze multiparameter large-change sensitivity in linear networks [7].

In this paper, we present a fast and accurate concurrent fault simulator CONCERT for nonlinear analog circuits. The paper is organized as follows. An overview of our fault simulation approach is discussed in Section 2. Section 3 provides the background for our fault simulation methodology. Section 4 presents the proposed concurrent fault simulation algorithm. Various techniques used for concurrent fault simulation are detailed in Sections 5, 6, and 7. The overall fault processing is discussed in Section 8. Simulation results are given in Section 9. This is followed by conclusions in Section 10.

2 Fault Simulation Methodology

The fault simulation methodology on which CONCERT is based on are as follows:

1. A set of training circuit instances, each instance corresponding to a set of different circuit parameter values, is first generated. This is performed using statistical methods so that the instances lie across and near the circuit specification boundaries, i.e. some of the circuit instances correspond to "good" circuits and some correspond to "bad" circuits. This set of training circuit instance is inserted into a fault list. The fault list is then expanded to include a fault universe which may include parametric faults and catastrophic short and open faults specified by the user.
2. For the specified transient stimulus, concurrent fault simulation is performed as follows: (a) between successive time steps all the circuits in the fault list are simulated before proceeding to the next time step; (b) if the circuit time step corresponds to one in which the CUT output(s) is sampled (the sampling frequency is an input to the simulator), then the measurement threshold for that time step is selected in such a way as to give unity yield coverage (this means that no "good" circuit instance is classified as "bad" by choice of the threshold); and (c) all circuit instances in the fault list that are "detected" due to the

* This work was sponsored by U.S. Defense Advanced Research Projects Agency (DARPA) under Contract No. F33615-95-2-5562

choice of the measurement thresholds are dropped and simulation is continued.

3. Fault coverage statistics that are tied to the circuit specifications are generated.

Note that here we differ from digital fault simulation in that: (i) the analog fault simulator need to select the measurement thresholds based on the specifications which may not given in the time domain; (ii) the "fault list" contains some "good" circuits as well as "bad" for the purpose of measurement threshold selection. In the above, if desired, fault dropping (2(c)) is not performed if the transient response of the CUT over the entire simulation interval for every fault is of interest, say, for diagnosis purposes.

The fault simulation methodology is illustrated in Figure 1. The key contributions of CONCERT are in step 2 described above. The simulator simulates all the entries in the fault list at every time step before proceeding to the next. This concurrent process allows CONCERT to *maximize the sharing of simulation effort for all circuit instances in the fault list*. The goal is to use as much information as possible from the simulation of every circuit instance in the fault list to simplify the simulation of the next. Three primary techniques is used to accomplish the concurrent fault simulation:

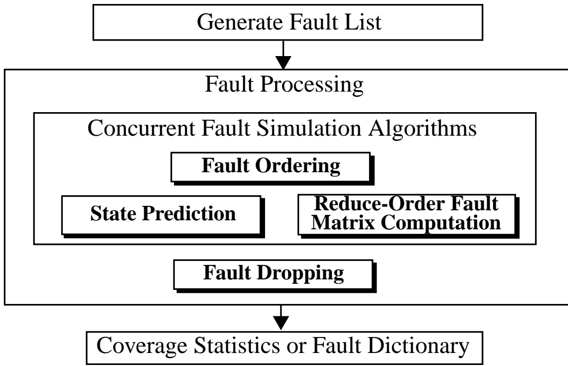


Figure 1 Fault Simulation Methodology of CONCERT

1. Fault Ordering: Based on the states of the circuit instances, all the circuit instances in the fault list are ordered at every time step. The dynamic ordering is done in such a way as to maximize the similarity between consecutive faults.
2. State Prediction: Given the order specified in the fault list, the state of the *i*-th circuit instance in the fault list is predicted from the state of the *i-1*-th simulated circuit instance in the fault list at every time step. The predicted state greatly reduces the number of Newton Raphson (NR) iterations for solving the system of nonlinear equations.
3. Reduced-order Fault Matrix (RFM) Computation: Based on the residual similarity between the nodal admittance matrices of the faulty and fault-free circuits, the system of faulty circuit equations is transformed into a reduced-order system of equations and solved with much less computational effort. Householder's formula [5] and sparse matrix technique [6] are used for the transformations.

The concurrent fault simulation techniques in CONCERT apply to DC, AC, and transient fault simulation of general linear and nonlinear analog circuits. For reasons of brevity, this paper will focus on DC and transient fault simulation algorithms.

3 Analog Circuit Simulation Basics

Our concurrent fault simulation approach is based on conventional modified nodal analysis (MNA) and numerical integration methods [4]. In the following, we first discuss the MNA formulation for circuit simulation. We will then show that most of the entries in the MNA matrix are invariant under fault and this greatly reduces the computations involved in solving the linearized system of equations.

In this paper, we use the term "*fault*" to denote a circuit instance in the fault list. The subscript $\boldsymbol{f}$ is used to denote a *fault*, the subscript $\boldsymbol{n}$ denotes the time step $\boldsymbol{t_n}$ in transient analysis, and the superscript $\boldsymbol{k}$ denotes the $\boldsymbol{k}$-th iteration in the NR equation solving procedure corresponding to time $\boldsymbol{t_n}$.

In general, the system of circuit equations is written as:

$$YU = I, \qquad Y \in \Re^{m \times m}, \; U, I \in \Re^{m} \tag{1}$$

where Y is the modified nodal admittance matrix of the circuit, U is the vector of *unknown node voltages and branch currents*, and I is the RHS contributed by *the known current and voltage sources*.

Consider the example nonlinear circuit in Figure 2, the linearized system of equations for DC analysis is:

$$\begin{bmatrix} g_d^k & -g_d^k & -1 \\ -g_d^k & g_d^k + \frac{1}{r} & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1^{k+1} \\ v_2^{k+1} \\ i_3^{k+1} \end{bmatrix} = \begin{bmatrix} -i_d^k \\ i_d^k \\ e \end{bmatrix} \tag{2}$$

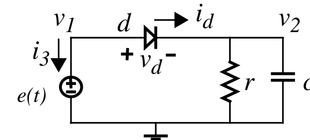


Figure 2 An example nonlinear circuit

where g_d^k is the dynamic conductance and i_d^k is the current of the diode, both evaluated at the diode's terminal voltage v_1^k - v_2^k. Starting with a DC input $e(0)$ and an *initial guess* v_1^0, v_2^0, and i_3^0, the system of equations is solved for v_1^{k+1}, v_2^{k+1}, and i_3^{k+1} iteratively for k=0, 1, ..., until the solution converges.

Transient analysis is based on stiffly stable integration methods with companion models for memory components [4]. For the circuit in Figure 2, the linearized equations at time t_n are:

$$\begin{bmatrix} g_d^k & -g_d^k & -1 \\ -g_d^k & g_d^k + \frac{1}{r} + y_n & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_{1_n}^{k+1} \\ v_{2_n}^{k+1} \\ i_{3_n}^{k+1} \end{bmatrix} = \begin{bmatrix} -i_d^k \\ i_d^k + j_n \\ e \end{bmatrix} \tag{3}$$

where y_n is the companion conductance and j_n is the companion current source of the capacitor corresponding to time t_n. Only after

the NR iterations converge at time t_n, can the simulation be advanced to the next time step in transient simulation.

In general, this can be formally stated as solving a system of non-linear circuit equations (at time t_n):

$$f(U_n) = 0 \tag{4}$$

using Newton-Raphson (NR) iteration method:

$$J(U_n^k) \cdot U_n^{k+1} = -f(U_n^k) + J(U_n^k) \cdot U_n^k \tag{5}$$

here $J(U_n^k)$ is the Jacobian matrix of $f(U_n)$ evaluated at U_n^k, and U_n^{k+1} is the k-th iterative solution. In each NR iteration, the nonlinear components need to be re-evaluated, i.e. the system of equations is re-linearized. The iterative solution is assumed to converge to the solution point when vectors U_n^k and U_n^{k+1} are significantly close. The number of NR iterations and its convergence heavily depend on the *initial guess* U_n^0. The cost of linearizing and solving the system of circuit equations dominate the computational cost of circuit simulation.

4 Concurrent Fault Simulation Algorithm

For transient fault simulation, all the faulty circuits along with the fault-free circuit are simulated concurrently at time t_n before simulation proceeds to the next time step. The same time step is used for all fault simulations and is determined by fault-free simulation.

```
Algorithm concurrentFaultSimulation(t_n)
01   normalCircuitSimulation(t_n, fault_free_circuit);
02   orderFaults(t_n, {fault_list});
03   precomputeRFM(t_n, fault_free_circuit);
04   for each f ∈ {fault_list} do
05        U_f^0 := predictState(f);
06        k := 0;
07        do                    //Newton-Raphson (NR) iterations
08          U_f^{k+1} := solveRFM(t_n, f);
09          k := k + 1;
10        until isConverge(t_n, f)
11    end for
```

Figure 3 Concurrent fault simulation algorithm

Figure 3 shows the algorithm of concurrent fault simulation at time t_n. The fault-free circuit is first simulated in function **normalCircuitSimulation()** which uses the conventional circuit simulation method [11]. All the faults is ordered in the fault list in function **orderFaults()**, which will be described in Section 7. Every fault is simulated using NR method as shown in the **do-until** loop. Functions **precomputeRFM()** prepares the common data for **solveRFM()**, which implements the RFM procedure for reducing the computational cost in solving the system of linearized equations. Function **predictState()** implement the state prediction method which reduce the number of NR iterations. The RFM procedure is explained in Section 5 and the state prediction method is explained in section 6. Function **isConverge()** checks if the NR iteration converges.

This general algorithm also applies to DC fault simulation, which is considered as a special case with t_n being 0. For transient fault simulation, the same procedure will be called at every time step for the entire test stimulus interval.

5 Reduced-order Fault Matrix Computation

When the circuit is faulty, the circuit equations are changed. But certain similarity between the faulty and fault-free circuit equations still exists. The difference in the circuit matrix can be captured by extracting the differences of the component conductances under that fault. Since the state of a nonlinear circuit is affected under fault, some nonlinear components in the circuit may also have different behavior than in the fault-free case.

It is important to point out that not all the nonlinear components' dynamic behaviors are affected by the fault at all times, especially in large mixed signal circuits. In the nonlinear circuit in Figure 2, only during a small period of time, the diode exhibits significantly different conductances in faulty and fault-free case. During majority part of its rectifier operation, the diode is "on" or "off", as illustrated in Figure 4 . During that period of transient simulation, the

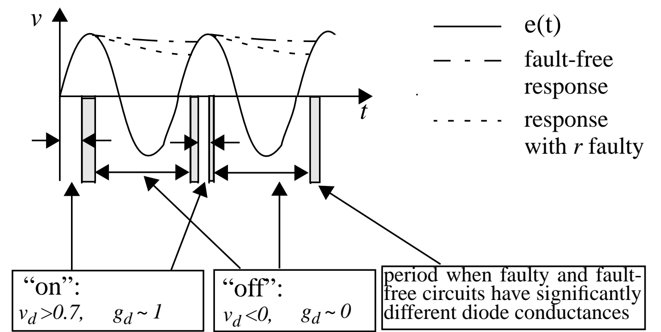


Figure 4 Similar responses and states of faulty and fault-free circuits in Figure 1

diode gives the same dynamic conductance as in the fault free case.

We define a component to be ***visible*** *if the difference between its dynamic conductances in faulty and fault-free circuits is larger than certain numerical threshold. Otherwise, it is* ***invisible****.*

A faulty component in a circuit is thus always *visible* according to this definition. The total number of *visible* components is equal to the number of faulty components plus the number of *visible* nonlinear components in the circuit. The *visibility* of a nonlinear component may change between NR iterations.

5.1 RFM Computation

Consider a faulty circuit with l *visible* components at k-th iteration, and the conductance differences between the fault-free and faulty case are δ_i^k, i=1, ... l, respectively. The difference matrix between the faulty and fault-free circuits can be expressed as:

$$\Delta Y_f^k = Y_f^k - Y = P_f D_f^k Q_f^T \tag{6}$$

where, $D_f^k = \begin{bmatrix} \delta_1^k & \dots & 0 \\ \dots & \dots & \dots \\ 0 & \dots & \delta_l^k \end{bmatrix} \in \Re^{l \times l}$, $P_f = \begin{matrix} r_1 \\ s_1 \\ \\ r_l \\ s_l \end{matrix} \begin{bmatrix} 1 & \dots & \dots \\ -1 & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & 1 \\ \dots & \dots & -1 \end{bmatrix}$, $Q_f = \begin{matrix} p_1 \\ q_1 \\ \\ p_l \\ q_l \end{matrix} \begin{bmatrix} 1 & \dots & \dots \\ -1 & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & 1 \\ \dots & \dots & -1 \end{bmatrix}$,

$P_f, Q_f \in \Re^{m \times l}$, and Y is the nodal admittance matrix of the fault-free circuit at its solution point. The i-th *visible* component is connected from node r_i to s_i, and controlled by the terminal voltage between node p_i and q_i. For the special case of a resistor, nodes (r_i,s_i) are the same as nodes (p_i,q_i).

The linearized faulty circuit equations can now be written as:

$$(Y + P_f D_f^k Q_f^T) \cdot U_f^{k+1} = I_f^k \tag{7}$$

By applying Householder's formula, we get:

$$U_f^{k+1} = (\mathbf{1} - Y^{-1} P_f [D_f^{k-1} + Q_f^T Y^{-1} P_f]^{-1} Q_f^T) Y^{-1} I_f^k \tag{8}$$

Since the LU factors of Y are known from fault-free simulation, it just takes one forward and backward substitution (FBS) to get an intermediate results of $E_f^k = Y^{-1} I_f^k$. And vector $F_f^k \equiv Q_f^T E_f^k$ can be easily filled up based on the simple matrix structure of Q_f

We define the Reduced-order Faulty Matrix (RFM) as $R_f^k \equiv [D_f^{k-1} + Q_f^T Y^{-1} P_f]$. If we know that the inverse of Y is Z, we can precompute part of the RFM:

$$Q_f^T Y^{-1} P_f = \begin{bmatrix} Z_{p_1 r_1} - Z_{p_1 s_1} - Z_{q_1 r_1} + Z_{q_1 s_1} & \dots & Z_{p_1 r_l} - Z_{p_1 s_l} - Z_{q_1 r_l} + Z_{q_1 s_l} \\ \dots & \dots & \dots \\ Z_{p_l r_1} - Z_{p_l s_1} - Z_{q_l r_1} + Z_{q_l s_1} & \dots & Z_{p_l r_l} - Z_{p_l s_l} - Z_{q_l r_l} + Z_{q_l s_l} \end{bmatrix} \tag{9}$$

Now, the faulty system of equations in (7) is *forward transformed* into a system of equations with reduced-order l:

$$R_f^k H_f^k = F_f^k, \qquad R_f^k \in \Re^{l \times l}, \; H_f^k, F_f^k \in \Re^l \tag{10}$$

Once we solve for H_f^k in (10), we need to *backward transform* the result to the original faulty system of equations (7), for which we need to find the matrix product:

$$G_f^k = Y^{-1} P_f H_f^k \tag{11}$$

There are two ways to compute this matrix product. One is based on the inverse matrix Z of Y, and the product comes from the matrix multiplication. Another way to find G_f^k is based on the observation that the column vector $P_f H_f^k \in \Re^m$ can be easily obtained from the simple matrix structure of P_f. Therefore, the system of equations in (11) can be solved using another FBS based on the LU factors of Y.

Finally, the solution of the k-th NR iteration is

$$U_f^{k+1} = E_f^k - G_f^k. \tag{12}$$

5.2 RFM Procedure and Its Complexity

This RFM computation is implemented in the procedure of **`solveRFM()`** in Figure 3. The procedure is illustrated in Figure 5. The decision of whether the RFM computation should be performed depends on l, the number of *visible* components. An important issue in circuit simulation is that the circuit matrix be sparse using sparse matrix techniques [10], and the complexity of solving equations (7) is practically $O(m^{1.5})$, where m is the order of the equations [10]. Since the RFM matrix is filled from the inverse of Y, it is not sparse. Hence the complexity of solving (10) is $O(l^3)$. Hence, we choose the decision function in Figure 5 as $T(m)=\sqrt{m}$. If l exceeds $\sqrt{m}$, the RFM approach is bypassed.

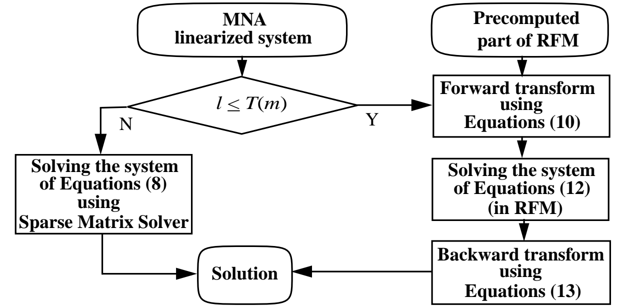


Figure 5 Procedure `solveRFM()`

Similar decision needs to make in backward transformation, since the computational complexity of FBS is around $O(m^{1.1})$ using sparse matrix techniques.

When there is only one *visible* component, which is the single faulty component, the faulty circuit behaves as a "linear circuit" during this iteration with respect to the fault-free circuit, since all the nonlinear components are *invisible*. Then, solving (10) needs just one division, and the backward transformation can also be simplified to just filling the vector G_f^k from Z.

The RFM procedure thus speeds up fault simulation in the following ways:

1. Only *visible* nonlinear components are evaluated at each iteration based on the differences in their terminal voltages.
2. Instead of the MNA matrix, a much smaller RFM is filled and factorized.
3. The cost of forward transformation may be shared between consecutive faults in the fault list, if they have very close circuit states, which result in the same RHS in equations (7).

It is important to note that all the entries in part of the RFM matrix in equations (9) are fault-independent and only depend on the Z matrix and the topological structure of the visible components. We can precompute this matrix from Z according to all the nonlinear and faulty components in the circuit. So the RFM matrix can be directly filled up based on the visible components at the k-th iteration. This pre-computation is performed in function **`precomputeRFM()`** in Figure 3. Since the LU factors of Y are readily available from the fault-free circuit simulation, inverting Y needs about two times more computations compared to one LU factorization. This computational overhead is well paid off when fault simulation for hundreds of faults are performed concurrently.

Experiments show that the RFM procedure is stable numerically. It can be shown that the RFM is not singular as long as the original MNA matrix is not singular.

6 State Prediction

While the RFM approach reduces the computational complexity within an NR iteration during fault simulation, *state prediction* method computes an initial guess close to the final state and reduces the number of NR iterations significantly. In this method, the similarities in response between two consecutive faults in the fault list under the same stimulus is exploited to compute the *best initial guess* for the NR iterations. Since the states of two consecutive faults are the closest according to fault ordering criteria, the state of the preceding faults in the fault list should be a good *initial guess* for the NR iterations in the simulation of the next fault. Since NR iterations converge quadratically near its solution point, a good initial guess will greatly reduce the number of NR iterations.

An intuitive method in state prediction for DC fault simulation of the next fault f_2 is to take the state of the preceding fault f_1 as the initial guess. That is:

$$U^0_{f_2} = U_{f_1} \tag{13}$$

This simple heuristic method works extremely well for DC fault simulation and results in much less number of NR iterations than other initial guesses based solely on the circuit structure. Further, since the initial guess is close to the final solution, the NR iterations for faulty circuits are now more likely to converge.

In the case of transient analysis, circuit simulators have used the state at the previous time step as an initial guess for NR iteration at the present time t_n. An i-th order polynomial using previous $i+1$ time step states is proposed for an initial guess [1]. In practice, most circuit simulators use a first order polynomial, since high-order polynomial interpolation offers about the same speedup.

Our state prediction approach for transient fault simulation is based on the similarity between the local response waveforms of two consecutive faults. To illustrate our approach, consider Figure 6

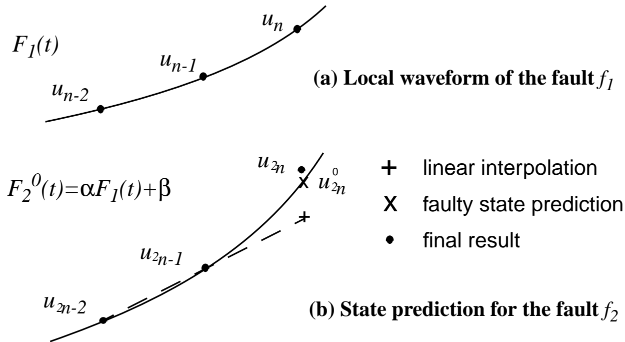


Figure 6 State prediction with reference to the preceeding fault

which shows the state prediction at time t_n for a fault f_2 using the simulation data from the preceding fault f_1. A second order polynomial function $F_1(t)$ is first built for the fault f_1, using the states, u_{n-2}, u_{n-1}, and u_n. This polynomial function is then used to build the prediction function for the next fault f_2:

$$F^0_2(t) = \alpha F_1(t) + \beta \tag{14}$$

which is also a 2nd order polynomial. To determine the constant α and β, two previous states u_{2n-1} and u_{2n-2} of fault f_2 are used.

In the case that a fixed time step is used for this example, with $t_n - t_{n-1} = t_{n-1} - t_{n-2}$, the value of the prediction function at time t_n can be derived as:

$$u^0_{2n} = f^0_2(t_n) = u_{2n-1} + \frac{u_n - u_{n-1}}{u_{n-1} - u_{n-2}}(u_{2n-1} - u_{2n-2}) \tag{15}$$

which will be used as the initial guess for solving u_{2n} in NR iteration for fault simulation. As we can see that the overhead of state prediction in fault simulation is very small.

7 Fault Ordering

Fault ordering is a key issue in predicting the state of the next fault accurately. The response of one fault may be more similar to the response of another fault rather than to that of the fault-free circuit. Therefore, more accurate state prediction can be achieved by using one fault response to predict the state of the next fault.

For precise DC fault simulation, all the faults in the fault list are ordered in terms of their parameter deviations, which are the only information accessible before DC simulation. A precise DC solution is necessary for transient fault simulation.

In transient fault simulation, all the faults in the fault list are ordered in terms of their previous transient responses as shown in Figure 7. When fault simulation proceeds to a new time step t_{n+1},

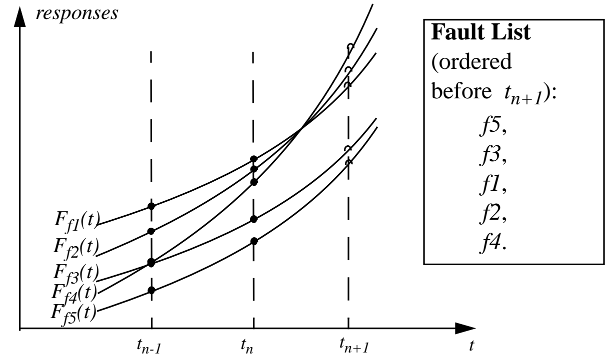


Figure 7 Fault ordering based on faulty responses

all the faults are ordered using the simulation data from the previous time steps. Then, fault simulation are performed at t_{n+1}. In particular, a weighted sum of the previous time output responses are used as a key λ for fault ordering, that is:

$$\lambda = a_0 u_{fn} + a_1 u_{fn-1} + \dots \tag{16}$$

Our experiments indicate that $a_0 = 2$, $a_1 = -1$ provides a good choice for this key function, which gives a fault ordering with better state prediction. Fault ordering is implemented in the procedure `orderFaults()` in Figure 3.

Compared to the complexity of solving a set of linear equations, the computational overhead of fault ordering and state prediction is negligible. At every time step, faults in the fault list are ordered, but few of the faults need to change their position with respect to the previous time ordering.

8 Fault Processing and Fault Coverage Analysis

In test generation applications, fault simulation is primarily used to estimate the fault or yield coverages of a test. For speeding up fault simulation, our approach employs early fault dropping to those faults that are detected at early steps of test stimulus application. Our approach also estimates thresholds to separate “good” and “bad” circuits.

8.1 Test Measurement Threshold

Thresholds on circuit specifications are often specified by circuit designer. For test other than specification tests (alternate tests), these thresholds have to be determined from simulation data.

In this paper, we propose concurrent Monte-Carlo simulation to compute thresholds for transient tests. The simulator first generates a large number L of training circuits with independent normal distributions of certain tolerance for the component parameters. Each training circuit is simulated to obtain its specifications. If the circuit satisfies all the circuit specifications, it is marked as “good”, otherwise it is marked as “bad”. After the specification simulation, we have a set of “good” circuits and a set of “bad” circuits. All lie across and near the circuit specification boundaries. Therefore, the transient test fault detection thresholds can be computed by simulating all the L circuits under the test stimulus, as shown in Figure 8. Assume that the test requires 100% yield coverage. Then, the

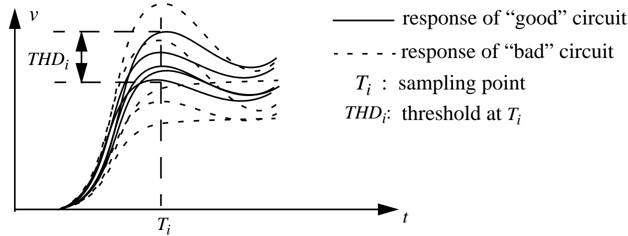


Figure 8 Calculation of measurement hreshold

threshold at a sampling point is the difference between the highest and the lowest responses of all the “good” circuits.

Figure 9 shows the algorithm of fault processing**.** The function

```
Algorithm Fault processing and coverage analysis
00 Input:  {netlist}, {fault_list}, {stimulus}, {sampling_times};
01 {training_circuits} := L statistical experiment circuits;
02 specsSimulation({training_circuits});
04 Insert {training_circuits} into {fault_list};
03 for each time step t_n do
09    concurrentFaultSimulation(t_n);
10    if t_n ⊂ {sampling_times} do                //fault dropping
11       THD := getThreshold(t_n, {training_circuits});
12       for each f_e ∈ {fault_list} do
13          if response(t_n,f_e) > THD
14             {fault_list} := {fault_list} − f_e;
15          end if
16       end for
17    end if
18 end for
19 reture {fault_list}  // the set of faults undetectable
```

Figure 9 Fault processing and coverage analysis

`specsSimulation()` performs the specification simulation for all the training circuits and marks the “good” and “bad” circuits, as described before. These training circuits are then inserted in the fault list. Thereafter, all the circuit instances in the fault list are simulated in `concurrentFaultSimulation()` described in Figure 3. If the current simulation time t_n is a sampling time for fault detection, the measurement threshold is computed in `getThreshold()` shown in Figure 8. All the faulty responses are checked against the threshold and those detected faults are dropped from the fault list. The algorithm ends up with a set of undetectable faults under this test stimulus. Therefore the fault coverage can be evaluated.

8.2 Early Fault Dropping

A typical transient test is illustrated in Figure 10. The response to a transient test for a CUT are sampled at certain time points. Once

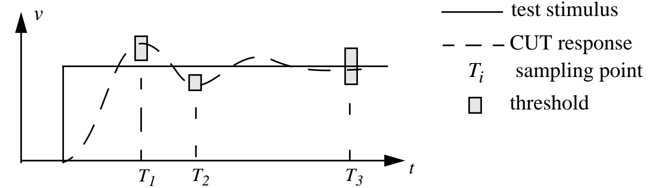


Figure 10 Transient test measurement thresholds

the response of a fault deviates from the expected fault-free response by a certain threshold at a sampling time, this fault is marked as detected. Therefore, this fault need not be simulated for the following sampling time points. In our concurrent fault simulation algorithm, we propose early fault dropping scheme to speed up the overall fault simulation. As the fault simulation proceeds to a new sampling time point, those faults which are detected are dropped from the fault list.

The fault simulation procedure obtains speedup through early fault dropping due to two reasons:

1. Since many faults are dropped during early sampling points, fewer faults need to be simulated per time step on an average.
2. Those faults which give larger response deviations at early sampling point are more expensive to simulate during later time steps, since large change in faulty states is harder to predict and takes more number of NR iterations during concurrent fault simulation. By dropping those faults, the average number of NR iterations for each faults is greatly reduced.

9 Experimental Results

The concurrent fault simulation algorithm has been implemented in a prototype simulation program called CONCERT. It uses SPICE level-1 models for devices like BJT, MOSFET, and diode etc. For solving the linearized circuit equations, CONCERT use the sparse matrix package developed in U.C. Berkeley [6].

For accuracy and speedup, we compare the performance of CONCERT with Spectre, an analog circuit simulator from Cadence Design Systems. Table 1 gives the various characteristics of the experimental circuits for evaluation. Among them, Biquad is a second-order low pass filter; Amp2 is a two-stage BJT amplifier; Slew is a slew rate filter; Front is a circuit constructed by combining Amp2 with Slew. U741 is the 741 opamp form CircuitSim90 benchmarks [18]. The opamps in Biquad, Slew, and Front are

Circuit name	# of faulty circuits	# of nodes	# of com-ponests	Test stimulus	
				signal	stop time
Biquad	170	20	58	Pulse(T_w=5ms)	10ms
Amp2	150	11	19	$50e\text{-}3\sin(2e4\pi t)$	0.2ms
Slew	200	10	37	$16\sin(20\pi t)$	140ms
Front	360	24	55	Pulse(T_w=0.1ms)	0.2ms
U741	120	25	38	$0.1\sin(2e4\pi t)$	0.4ms

Table 1 Example circuit characteristic

described in macromodel which includes input and output voltage limiting diodes linearized at v_d>0.7volt.

Each faulty circuit in the fault list is generated by injecting single catastrophic or parametric fault associated with a linear component in the circuit. Two catastrophic faults (short and open) and 8 parametric faults (with 5%, 15%, 50%, 80%, 120%, 150%, 200%, and 1000% of the nominal value, respectively) are generated corresponding to each linear component.

Circuit name	# of faults	DC fault simulation			TR fault simulation		
		Spectre (sec)	Concert (sec)	Speed up	Spectre (sec)	Concert (sec)	Speed up
Biquad	170	1.7	0.01	**170**	57.8	0.46	**125**
Amp2	150	3	0.03	**100**	15.0	2.63	**5.7**
Slew	200	2	0.01	**200**	38.0	3.04	**12.5**
Front	360	10	0.12	**90**	115	12.8	**9.0**
U741	120	6	0.26	**23**	25.4	10.2	**2.5**

Table 2 Speedup of Concert over Spectre in simulation for the fault-free and all the faulty circuits

Table 2 Compares the DC and transient (TR) fault simulation CPU time for the example circuits using Spectre and CONCERT on a Sun Ultra1. The data for Spectre is the intrinsic simulation time reported by Spectre accumulated for all faults. The actual CPU time for Spectre is even longer due to the overhead involved in setting up MNA. In transient fault simulation of the Biquad filter, two order of magnitudes speedup was obtained using CONCERT. Even for U741 with precise device modeling for its 23 BJTs, we still get 2.5 times speedup. For all other circuits, CONCERT was 6-12 times faster than Spectre. In DC fault simulation, higher speedups are obtained than in transient fault simulation. This is due to the fact that DC fault effects are much more localized, and that the state prediction method are far more better than random initial guess which is the case in DC circuit simulation.

We also collected some statistical data during fault simulation to show how speedup is achieved in CONCERT and why different speed up are obtained for various types of circuits. Table 3 gives the average number of NR iterations per time step per fault, with and without using the fault state prediction technique. It can be seen that the number of iterations is reduced by 50-165% in CONCERT using state prediction for transient fault simulation, and much more for DC case.

Further speedup is obtained by using the RFM technique, which gives quite different speedup for different type of circuits. Table 4

Circuit name	DC fault simulation		TR fault simulation	
	without state prediction	with state prediction	without state prediction	with state prediction
Biquad	5	1	2	1
Amp2	21	3.58	2.87	1.91
Slew	11	1	3.93	1.48
Front	24	1.8	3.23	1.89
U741	55	6.2	4.23	2.31

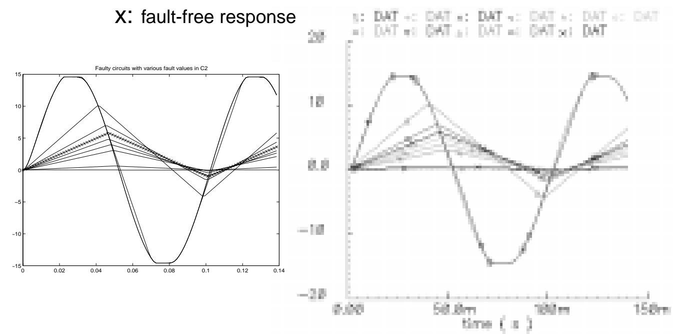
Table 3 Average number of NR iterations

Circuit name	# of nodes	DC RFM	TR RFM
Biquad	20	1	1
Amp2	11	3.73	4.11
Slew	10	1	2.35
Front	24	2.27	4.64
U741	25	14.5	15.9

Table 4 Average order of RFM

shows the average order of the RFM for all the faulty circuit simulations performed concurrently in DC and TR fault simulation. The order of RFM is equal to the number of *visible* components at any simulation time, since only *visible* components will contribute to the RFM. The order of RFM for the Biquad filter is unity at every time step. This is due to the fact that the circuit is operating in its linear range under the test stimulus and CONCERT is very efficient when it detects that the order of RFM is one. For the other circuits, the order of RFM is higher. Among the nonlinear circuits, the fault simulation for the slew rate filter gets the largest speedup because most of its nonlinear components are voltage limiting diodes and the average number of visible components is very small. Therefore, the speedup obtained by CONCERT is mainly related to the average number of visible components in fault simulation. Circuits described in marcomodels or behavioral models gets more speedup in fault simulation because of the less number of visible components.

The output waveforms generated from CONCERT were found to match with that from Spectre simulation, with less than 5% maximum error which is mainly due to the slight difference in time steps and device modeling. Figure 11 shows the simulation outputs from CONCERT and Spectre for the slew rate filter, with a test stimulus of $16\sin(20\pi t)$. About 100 time steps are simulated both in CONCERT and Spectre.



(a) CONCERT outputs **(b) Spectre outputs**

Figure 11 Simulation output of one fault-free and ten faulty circuits due to the capacitor C2 in the slew rate filter

To demonstrate the fault processing capabilities of CONCERT, we use the stimuli given in Table 1. We consider five measurements made on the transient response of the circuit excited by these stimuli. The sampling are distributed evenly on the time axis. The results of training set simulation is given in Table 5. The training set consists of 100 circuits with multiple parametric faults. This shows that CONCERT can perform fault simulation of multiple faults efficiently. The supporting simulation statistics are also shown in the table. The simulation time in Table 5 are the CPU time for Spectre and Concert, both include the overhead of setting up the MNA in simulation.

The fault coverage of the transient measurements was estimated using the fault list given in Table 1. The results are summarized in Table 6. The simulation time for Spectre are copied form Table 2. We can see that CONCERT achieves considerable speedup for the purpose of fault coverage evaluation.

Circuit name	Spectre (sec)	Concert (sec)	Speed up	ave. # of iteration	ave. order of RFM	fault coverage
Biquad	57.8	0.27	**214**	1	1	81%
Amp2	15.0	1.18	**12.7**	1.29	3.51	87%
Slew	38.0	1.49	**25.5**	1.25	1.70	75%
Front	115	8.27	**13.9**	1.45	3.79	64%
U741	25.4	2.87	**8.9**	1.43	12.6	84%

Table 6 Speedup of Concert over Spectre in fault simulation for fault coverage evaluation of the test

Circuit name	# of training circuit	Spectre (sec)	Concert (sec)	Speed up	ave. # of iteration
Biquad	100	65.0	13.4	**4.85**	1
Amp2	100	27.0	4.42	**6.11**	1.39
Slew	100	37.0	5.57	**6.64**	1.21
Front	100	50.0	13.7	**3.65**	1.48
U741	100	40.0	7.55	**5.30**	1.57

Table 5 Speedup of Concert over Spectre in fault simulation for training circuits

10 Conclusions

We present novel concurrent fault simulation algorithms for nonlinear analog circuits. We believe that CONCERT is the first of its kind of concurrent fault simulator for nonlinear analog circuits. Significant fault simulation speedup is obtained for highly nonlinear circuits with pure analog signals, without any loss of accuracy. Higher relative speedup are obtained for circuits described in macromodels, which are more attractive for dealing with complex mixed-signal circuits.

Future investigation includes how to achieve further speedup at the expense of accuracy. For example, how to reduce the number of *visible* components by reducing the visibility threshold. Other techniques for speeding up circuit simulation, such as multi-rate simulation, event driven circuit simulation and use of piecewise linear models, can be used in conjunction with our concurrent simulation approach to further speedup fault simulation for various types of electronic circuits.

11 Acknowledgments

The author would like to thank Ramakrishna Voorakaranam and Pramodchandran N Variyam for help and valuable discussions.

12 References

[1] R.K. Brayton, F.G. Gustavson, G.D. Hachtel, "A new efficient algorithm for solving differential-algebraic systems using implicit backward-differentiation formulas," *Proceedings of the IEEE*, Vol. 60, No. 1, pp. 98-108, Jan. 1972

[2] G. Devarayanadurg, P. Goteti and M. Soma, "Hierarchy based statistical fault simulation of mixed-signal ICs," *Proceedings, International Test Conference*, pp. 521-527, 1996.

[3] R. J. A. Harvey, "Analogue fault simulation based on layout dependent fault models," *Proceedings, International Test Conference*, pp. 641-649, 1994.

[4] C.W. Ho, A.E. Ruehli and P.A. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuits Syst*. Vol.CAS-22, pp. 504-509, June 1975.

[5] A. S. Householder, "A survey of some closed methods for inverting matrices," *SIAM J. Appl. Math*., Vol. 5, No. 3, pp. 155-169, 1957.

[6] K.S. Kundert, "Sparse matrix techniques," in *Circuit Analysis, Simulation and Design*, A.E. Ruehli(editor), North-Holland, 1986.

[7] K. Leung and R. Spence, "Multiparameter large-change sensitivity analysis and systematic explorations," *IEEE Trans. Circuits Syst*., Vol. CAS-22, pp. 15-21, Jan. 1975.

[8] L.W. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, PhD thesis, EECS Dept., Univ. Calif., Berkeley, May 1975, Memorandum no. ERL-M520.

[9] N. Nagi, A. Chatterjee and J. A. Abraham, "DRAFTS: Discretized analog circuits fault simulation," *Proceedings, IEEE/ACM Design Automation Conference*, pp. 509-514, June 1993.

[10] A.R. Newton, A. Sangiovanni-Vincentelli, "Relaxation-based circuit simulation," *IEEE Trans. on Elec. Dev.*, Vol. ED-30, No.9, pp.1184-1207, Sept. 1983

[11] L.T. Pillage, R.A. Rohrer, C. Visweswariah, *Electronic Circuit and System Simulation Methods*, McGraw-Hill, 1995

[12] E.W. Thompson and S.A. Szygenda, "Digital logic simulation in a time-based, table-driven environment part 2, parallel fault simulation," *Comput*., Vol. 8, pp. 38-49, Mar, 1975.

[13] M.W. Tian and C.J.R. Shi, "Rapid frequency-domain analog fault simulation under parameter tolerances," *Proceedings, IEEE/ACM Design Automation Conference*, pp. 275-280, June 1997.

[14] E.G. Ulrich and T. Baker, "Concurrent simulation of nearly identical digital networks," *Comput*., Vol. 7, pp. 39-44, Apr. 1974.

[15] P. Variyam, A. Chatterjee, "FLYER: Fast fault simulation of linear analog circuits using polynomial waveform and perturbed state representation," *VLSI DESIGN'97*, pp. 408-412, Jan. 1997.

[16] R. Voorakaranam, A. Gomes, S. Cherubal, A. Chatterjee, "Hierarchical fault simulation of feedback embedded analog circuits with approximately linear to quadratic speedup," *Proc. International Mixed Signal Testing Workshop, 1997*.

[17] M. Zwolinski, A.D. Brown, C.D. Chalk, "Concurrent Analogue Fault Simulation," *Proc. International Mixed Signal Testing Workshop*, pp.42-47 1997.

[18] "CircuitSim90", *1990 Circuit Simulation and Modeling Workshop at MCNC*, http://www.cbl.ncsu.edu/CBL_Docs/csim90.html.