ftd: An Exact Frequency to Time Domain Conversion for Reduced Order RLC Interconnect Models†

Ying Liu, Lawrence T. Pileggi, Andrzej J. Strojwas
Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213

Abstract

Recursive convolution provides an exact solution for interfacing reduced-order frequency domain representations with discrete time domain models of piecewise linear voltage waveforms. The state-space method is more efficient, but not exact, and can sometimes produce large time domain errors. This paper presents a new algorithm, ***ftd*** *(frequency to time domain), for incorporating linear frequency domain macromodels into time domain simulators.* ***ftd*** *provides accuracy equivalent to recursive convolution with efficiency that is superior to the state-space methods.*

1: Introduction

There are two popular methods for converting a frequency domain model into a discrete time domain model. The first is recursive convolution[1], where the input is assumed to be piece-wise linear and decomposed as different ramps at each time point. This permits the convolution to be carried out in a recursive way. The second approach is the state-space method[2], whereby a numerical integration scheme such as trapezoidal is applied to the state space realization of the frequency domain model to compute a discrete time domain model. Among the two, recursive convolution achieves the best possible accuracy[3], but can be inefficient due to the inherent complexity of the algorithm. In contrast, the state-space method requires fewer floating-point operations at each time point than recursive convolution, but has inferior accuracy due to the required numerical integration scheme. Moreover, both methods are theoretically complicated and, therefore, somewhat difficult to implement. This difficulty not only impacts their widespread use, but also makes it difficult to optimize the software routines, which indirectly impacts the runtime efficiency.

† This work was supported, in part, by the Semiconductor Research Corporation under Contract DC-068.067 and SGS-Thomson Microelectronics.

DAC 98, June 15-19, 1998, San Francisco, CA USA
ISBN 1-58113-049-x/98/06…$5.00

In this paper, we propose an efficient and accurate algorithm, called ***ftd***, for converting a frequency domain model to a discrete time domain model for circuit and timing simulation. ***ftd*** uses an efficient algorithm for direct computations of the non-zero state response at each time point. ***ftd*** provides accuracy equivalent to that of the recursive convolution but with the same complexity as the state-space method. Furthermore, due to the algorithmic simplicity of ***ftd***, empirical results demonstrate that its implementation is more efficient than the state-space method too.

2: Background

For timing simulation, a linear interconnect subcircuit can be pre-processed into a reduced-order **Y** matrix[4]. A two-port example is written as:

$$\begin{bmatrix} I_1(s) \\ I_2(s) \end{bmatrix} = \begin{bmatrix} y_{11}(s) & y_{12}(s) \\ y_{21}(s) & y_{22}(s) \end{bmatrix} \cdot \begin{bmatrix} V_1(s) \\ V_2(s) \end{bmatrix} \quad (1)$$

The first equation in (1) can be expressed as:

$$I_1(s) = y_{11}(s) \cdot V_1(s) + y_{12}(s) \cdot V_2(s) \quad (2)$$

where the general form of any $y_{ij}(s)$ term is:

$$y(s) = \frac{b_{q-1}s^{q-1} + \ldots + b_1 s + b_0}{s^q + a_{q-1}s^{q-1} + \ldots a_1 s + a_0} \quad (3)$$

Using recursive convolution[1], state space method[2], or the approach proposed in this paper, the discrete time domain model of (2) at time point t_k can be written as:

$$i_1(t_k) = g_{11}(t_k)v_1(t_k) + g_{12}(t_k)v_2(t_k) + I_{1eq}(t_k) \quad (4)$$

once a time step is chosen. It follows from (1) that:

$$\begin{bmatrix} i_1(t_k) \\ i_2(t_k) \end{bmatrix} = \begin{bmatrix} g_{11}(t_k) & g_{12}(t_k) \\ g_{21}(t_k) & g_{22}(t_k) \end{bmatrix} \cdot \begin{bmatrix} v_1(t_k) \\ v_2(t_k) \end{bmatrix} + \begin{bmatrix} I_{1eq}(t_k) \\ I_{2eq}(t_k) \end{bmatrix} \quad (5)$$

The equivalent circuit realization of (5) is shown in Fig.1, where each port includes an equivalent conductance, current source, and VCCS. During the simulation process, at each time point, these discrete time domain models are constructed based on a reduced-order frequency domain approximation. This N-port model is combined with the models connected to its ports via Modified Nodal Analysis(MNA). The solution of the MNA equations along with

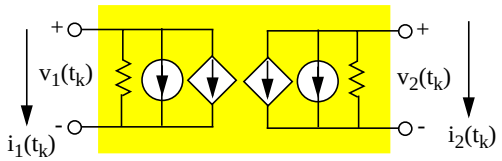

FIGURE 1: Discrete time domain model of a two-port.

the non-linear elements will yield the overall circuit solution.

It can be shown that for all three methods the values of conductances and VCCS's depend only on the time step. Therefore, when the time step is fixed for consecutive time points, these parameter values are constant. The equivalent current sources are always updated at each time point, thereby dominating the runtime for each method. For this reason we will focus on the updating procedure of the equivalent current source models when comparing the algorithmic complexity of the three methods.

In recursive convolution, it is assumed that the input voltage is piecewise linear(PWL) as multiple incremental ramps occur at each time point with different slopes. The overall response can be calculated as the sum of multiple zero-state ramp responses up to the present time point. By careful bookkeeping of these variables, the calculation can be carried out in a recursive and efficient way. A complete derivation of recursive convolution can be found in [1][7]. Recursive convolution is exact for PWL voltages, however this accuracy is achieved at the expense of efficiency. For an N-port with a q-th order approximation, the number of internal variables that are updated at each time point is $3N^2q$. The algorithm complexity is estimated as $O(7N^2q)$ at each time point. When the time step is changed, updating of the parameters requires $O(2N^2q)$ exponential evaluations and $O(4N^2q)$ floating-point multiplications/divisions.

In the state-space method, the admittances in the form of (3) are re-formatted into a controllable canonical state-space form[5]. Numerical integration such as the trapezoidal method is used to calculate the response. A complete description of the state-space method can be found in [2][7]. For the state-space method, the number of variables that must be updated at each time point is N^2q. The computation requires $O(q^2N^2)$ of floating-point multiplications/divisions. Because a numerical integration scheme is used, no exponential evaluation is needed. Since it requires fewer floating-point operations at each time point, the state-space method is faster than recursive convolution for a fixed time step. However, since a numerical integration scheme is used, the state-space method is not as accurate as recursive convolution.

3: Description of *ftd*

In ***ftd***, each admittance of the form given by (3) is first decomposed into a sum of partial fractions. This pre-processing step represents some computational overhead for ***ftd*** and recursive convolution. However, most model order reduction schemes rely on calculation of the poles and residues to verify accuracy and stability.

The decomposition can be formulated in matrix notation as:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}v(t) \\ i(t) = \mathbf{C}\mathbf{x}(t) \end{cases} \tag{6}$$

with

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_q \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} p_1 & 0 & \dots & 0 \\ 0 & p_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & p_q \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} k_1 & \dots & k_q \end{bmatrix}$$

Unlike recursive convolution, ***ftd*** uses directly the **exact** solution of the differential equation (6), which can be written as[5]:

$$\mathbf{x}(t) = e^{\mathbf{A}t} \cdot \mathbf{x}(0) + e^{\mathbf{A}t} \int_0^t e^{-\mathbf{A}\tau} \mathbf{B} v(\tau) d\tau$$

Assuming that we know the solution at the time point t_{k-1}, then the exact solution at time point t_k can be expressed as a zero-input response term plus a zero-state response term:

$$\mathbf{x}(t_k) = e^{\mathbf{A}\Delta t_k} \cdot \mathbf{x}(t_{k-1}) + \int_0^{\Delta t_k} e^{\mathbf{A}\tau} \mathbf{B} v(t_k - \tau) d\tau \tag{7}$$

Since at the time point t_{k-1} we don't know the voltage response shape to the time point t_k, the integral in (7) cannot be computed explicitly. However, as with the recursive convolution, and as generally assumed in circuit and timing simulation, for a reasonably small timestep we can assume that the waveshape is piecewise linear. Thus, the voltage between the time points t_{k-1} and t_k is expressed as follows:

$$v(t_k - \tau) = v(t_k) - \left(\tau \cdot \frac{v(t_k) - v(t_{k-1})}{\Delta t_k} \right) \quad for \quad 0 \le \tau \le \Delta t_k \tag{8}$$

Using the piecewise linear assumption in (8), the integral in (7) can be evaluated explicitly

$$\mathbf{x}(t_k) = e^{\mathbf{A}\Delta t_k} \mathbf{x}(t_{k-1}) + \left(\mathbf{A}^{-1} e^{\mathbf{A}\Delta t_k} \mathbf{B} - \frac{\mathbf{A}^{-2}}{\Delta t_k} (e^{\mathbf{A}\Delta t_k} - \mathbf{I}) \mathbf{B} \right) \cdot$$

$$v(t_{k-1}) + \left(\frac{\mathbf{A}^{-2}}{\Delta t_k} (e^{\mathbf{A}\Delta t_k} - \mathbf{I}) \mathbf{B} - \mathbf{A}^{-1} \mathbf{B} \right) v(t_k)$$

This formula appears to be formidable. For a general realization of {**A**, **B**, **C**}, it is quite difficult to carry out the above matrix operations. But since we expand the admittance terms into a sum of partial fractions, the **A** matrix is diagonal and all of the entries in **B** are equal to one. Therefore, the matrix operations in the above equation are actu-

ally scalar operations. After some simplification, the equation can be re-written as:

$$x_i(t_k) = \zeta_i(\Delta t_k) \cdot x_i(t_{k-1}) + \eta_i(\Delta t_k) \cdot v(t_{k-1}) + \theta_i(\Delta t_k) \cdot v(t_k)$$
$$for \quad i = 1, 2, \ldots, q$$

with the parameters ζ, η and θ defined as:

$$\zeta_i(\Delta t_k) = e^{p_i \Delta t_k}$$
$$\eta_i(\Delta t_k) = \frac{e^{p_i \Delta t_k}(p_i \Delta t_k - 1) + 1}{\Delta t_k \cdot p_i^2} \tag{9}$$
$$\theta_i(\Delta t_k) = \frac{1}{\Delta t_k}\left(-\frac{1}{p_i^2} - \frac{1}{p_i}\Delta t_k + \frac{1}{p_i^2} e^{p_i \Delta t_k}\right)$$

These parameters depend only on the poles and the time step. They remain constant when the time step is fixed for consecutive time points.

The current at the time point t_k can be expressed as:

$$i(t_k) = I_{eq}(t_k) + g(t_k) \cdot v(t_k)$$

where the equivalent current source and the equivalent conductance are defined as:

$$g(t_k) = \sum_{i=1}^{q} k_i \cdot \theta_i$$
$$I_{eq}(t_k) = \left(\sum_{i=1}^{q} k_i \cdot \eta_i\right) v(t_{k-1}) + \sum_{i=1}^{q} k_i \zeta_i x_i(t_{k-1}) \tag{10}$$

Once the voltage at time point t_k is known, the states should be updated to prepare for the next time point calculation.

When there are complex poles in the admittance, ***ftd*** involves the evaluation of complex numbers. Since the complex numbers cost twice as much in storage, they should be avoided whenever possible. The discussion of the complex pole algorithm can be found in [7].

4: Implementation for Multiports

ftd is very simple to implement. Following the 2-port example shown in Section 2, the goal is to calculate the discrete time-domain model of a two-port shown in (5) at each time point. We use superscripts to distinguish between different ports and subscripts to distinguish between different poles and residues for the four admittance terms. Upon choosing the appropriate time step, the parameters ζ, η, θ for each real

The number of states to update in ***ftd*** is 4q, which can be stored in a single vector **x**. At each time point, if the time step is unchanged, only the equivalent current sources have to be updated. The equivalent current source updating algorithm can be carried out as shown in Fig.2. Note that all the floating-point operations involved are SAXPY ($y \leftarrow a \cdot x + y$) type of operation. For a modern computer architecture, this can be carried out with the utmost efficiency.

At each time point:

◆Calculate intermediate variables, store them in vector **x**:

$$x_i^{(mn)} \leftarrow \zeta_i^{(mn)} \cdot x_i^{(mn)} + \eta_i^{(mn)} \cdot v_n(t_{k-1})$$

◆Calculate the equivalent current sources:

$$\begin{bmatrix} I_{1eq} \\ I_{2eq} \end{bmatrix} \leftarrow \begin{matrix} \sum_{i=1}^{q} k_i^{(11)} \cdot x_i^{(11)} + \sum_{i=1}^{q} k_i^{(12)} \cdot x_i^{(12)} \\ \sum_{i=1}^{q} k_i^{(21)} \cdot x_i^{(21)} + \sum_{i=1}^{q} k_i^{(22)} \cdot x_i^{(22)} \end{matrix}$$

◆Stamp equivalent conductances and current sources into MNA matrix and solve MNA matrix for the voltages.

◆Update the state vector for the next time point.

$$x_i^{(mn)} \leftarrow x_i^{(mn)} + \theta_i^{(mn)} \cdot v_n(t_k)$$

FIGURE 2: Flowchart of equivalent current source updating scheme.

5: LTE and Complexity of the Algorithms

Following the common assumption of LTE(Local Truncation Error) estimation, we assume that the exact value of the state at the time point t_{k-1} is known. Furthermore, we assume that the voltages are perfectly linear between time points t_k and t_{k-1}. A complete discussion of the LTE of the three methods can be found in [7]. The results are tabulated in Table 1.

We compare the complexity for a fixed time step in terms of the floating-point operations. The second complexity measure considers the number of floating-point operations when the time step changes. Details can be found in [7], but the results are summarized in Table 1.

	Recursive Convolution	State-Space Method	*ftd* Method
# of variables	$3N^2q$	N^2q	N^2q
Complexity[1]	$O(7N^2q)$	$O(3N^2q)$	$O(3N^2q)$
Complexity[2]	exp: $O(2N^2q)$ mul: $O(5N^2q)$	exp: none mul: $O(N^2q^2)$	exp: $O(N^2q)$ mul:$O(5N^2q)$
LTE	0	$p\Delta t_k^2(v_k - v_{k-1})$	0

TABLE 1: LTE and estimation of complexity for all three methods.

It is apparent that the recursive convolution requires the largest amount of floating-point computation for a fixed time step analysis. At first glance it would appear that the state-space method and ***ftd*** have identical complexity when the time step is fixed, however, for the state-space method the state $x_2(t_k)$ cannot be calculated until $x_1(t_k)$ is available. Similarly, $x_3(t_k)$ cannot be calculated until $x_2(t_k)$ is available, and so on. ***ftd*** does not suffer from this dependency, therefore with modern computers, and pipeline processing, the ***ftd*** updates will run much more efficiently.

When the time step changes, the state-space method requires the largest amount of multiplications/divisions, but avoids exponential evaluations. Moreover, when there are complex poles, both recursive convolution and ***ftd*** also require evaluation of triangular functions. In practice, the exponential and triangular functions can be stored in a look-up table. As a result, the recursive convolution and ***ftd*** may spend less time in re-calculating the parameters than the state-space method when the time step is changed.

6: Results

The first example is a clock tree with 64 leaf nodes. Each interconnect segment is modeled as a lumped RC segment providing a 65-port circuit. AWE[4] is used to calculate a 4th order model for all admittance terms. The resulting **Y**(s) macromodel is a 65×65 matrix. We compare recursive convolution, the state space method, and ***ftd*** in terms of a Matlab implementation on an IBM RS/6000 43P-140.

We plot the absolute errors as a function of time for one port node in Fig.2 by comparing to a SPICE simulation with a 0.1 picosecond timestep. The error plots are shown for two different timesteps. When the time step is 0.2 picosecond, the three methods have comparable error. However, when the time step grows to 1 picosecond, the state-space error is considerably larger than the other two methods.

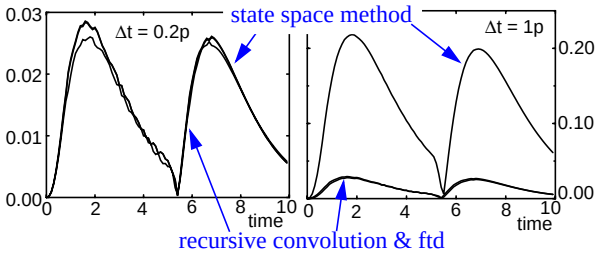

FIGURE 2: The errors for the three methods using two different time steps. Note the different plot scales.

Table 2 shows the runtimes for the three methods(without the use of table models for exponential function calls). As expected, ***ftd*** is the most efficient, but these results must be carefully interpreted. First of all, Matlab is an interpretive language, thus the running time calculation is inaccurate. Secondly, Matlab is optimized to execute vector/matrix operations, but the optimization relies on skillful programming. For the recursive convolution implementation, the complicated updating algorithm was by far the more straightforward to implement, however, the serial dependency discussed in Section 5 does not allow for certain optimizations and slows down the simulation. We must emphasize, however, that the authors spent a significant amount of time in implementing and optimizing all three methods as much as possible. But as indicated above, the simplicity of the ***ftd*** algorithm can lead to better runtime efficiency because it is more easily optimized for a wide variety of computer platforms.

Time Step	Recursive Convolution	State Space Method	Our Method
Δt = 0.2p	291.90 sec	274.99 sec	141.49 sec
Δt = 1p	60.98 sec	54.52 sec	30.36 sec

TABLE 2: Simulation time of the three methods.

In the second example, ***ftd*** was used with TETA(Transistor-level Engine for Timing Analysis)[6] to simulate a simple coupled RLC line problem with SPICE-like nonlinear drivers. The loads at the far ends of the lumped transmission model are linear capacitors, as shown in Fig.3. The interconnect is a six-port with six identical nonlinear drivers. The reduced-order model of the two-port is calculated using AWE to obtain a 3rd order approximation. The simulation result is shown in Fig.3 for a timestep of 1 picosecond. The SPICE and ***ftd*** (using TETA for the nonlinear elements) plots are indistinguishable.

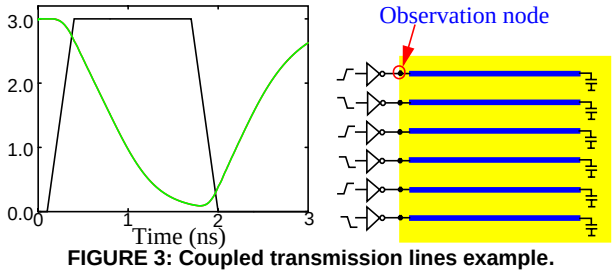

FIGURE 3: Coupled transmission lines example.

7: Conclusion

This paper presents a new algorithm for the converting frequency domain macromodels into the discrete time-domain models for circuit and timing simulation. Both the theoretical analysis and numerical results indicate that ***ftd*** achieves the highest possible accuracy while requiring the least computational resources. An important feature of ***ftd*** is that it is very straight-forward and can be easily implemented with increased efficiency on most modern hardware platforms.

References

[1]J. E. Bracken, V. Raghavan and R. A. Rohrer, “Interconnect Simulation with Asymptotic Waveform Evaluation”, IEEE Trans. on Circuits and Systems, vol. 39, No. 11, Nov. 1992.

[2]S.-Y. Kim, N. Gopal and L. T. Pillage, “Time-Domain Macromodels for VLSI Interconnect Analysis”, IEEE Trans. on CAD, vol. 13, No. 10, Oct. 1994.

[3]T. V. Nguyen, “Efficient Simulation of Lossy and Dispersive Transmission Lines”, IEEE/ACM Proc. DAC, 1994.

[4]L. T. Pillage and R. A. Rohrer, “Asymptotic Waveform Evaluation for Timing Analysis”, IEEE Trans. on CAD, vol. 9, no. 4, Apr. 1990.

[5]C.-T. Chen, “Linear System Theory and Design”, Holt, Rinehart and Winston, Inc., 1984.

[6]F. Dartu and L.T. Pileggi, “TETA: Transistor-Level Engine for Timing Analysis”, IEEE/ACM Proc. DAC, Jun. 1998.

[7]Y. Liu, L.T. Pileggi and A.J. Strojwas, “ftd: A Frequency to Time Domain Conversion Algorithm for Interconnect Simulation”, submitted to IEEE Tran. CAD.