# A New Algorithm for Accurate Wire Endpoint Delay Estimation

Zhao Wang[1], Carl Sechen[2]

The University of Texas at Dallas, TX, 75080, U.S

[1]wangzhao1984@hotmail.com; [2]carl.sechen@utdallas.edu

***Abstract*-This paper proposes a new estimation algorithm that accurately predicts the delay, slew rate at each wire endpoint based on an RC extraction, regardless of the number of RC sections and the effective capacitance (Ceff) seen by a gate. In addition, an improved slew rate estimation metric based on prior work is introduced. The delay and edge rate results compare favourably with respect to HSPICE simulation results and are more accurate than those from the leading commercial static timing analysis tool.**

***Keywords- Delay Estimation; RC Networks; Equivalent Resistor***

## I. INTRODUCTION

The propagation delays of interconnect lines, and particularly the delays to specific wire endpoints, play an increasingly important role for today's submicron technologies. The chip sizes and wire lengths have not reduced much due to current elaborate levels of integration, while interconnect wires have become narrower, thicker and more tightly packed. For quite some time it was sufficient to consider only the effects of interconnect capacitance when performing static timing analysis (STA). However, with current technologies, the impact of wire resistances on gate and path delays is critically important for many wires. Thus, an accurate gate-plus-wire delay model is necessary in order to in turn achieve accurate STA results.

A variety of wire-modeling methods have been proposed, including the lumped capacitance model, pi-model (or π-model), lumped RLC network model, lumped RC network model, and distributed transmission line model [1]. The lumped capacitance model (Fig. 1) simply sums up all extracted capacitances, ignoring resistances and inductances, saving computation time but underestimating path delays. The single π -model (Fig. 2) contains two resistors and one capacitor (in a π configuration). Instead of analyzing the entire network, the π -model only converts the network to a single π -structure and analyzes it [2, 3]. This is the simplest model that captures some of the effects of resistive shielding. The distributed transmission line model (a lumped version of this is shown in Fig. 3) fully represents the interconnect, including all resistor, capacitor and inductor effects, yielding good accuracy. However, this model is usually overly complex, leading to unacceptable computation times, unless the signal ramp time, and the corresponding propagation distance for that duration, approaches the length of a portion of the net [1, 3]. The lumped RC network model shown in Fig. 4 is currently the mostly widely used model in industry for STA, where it is specified in a Standard Parasitic Exchange Format (SPEF) file. The lumped RC model normally uses a number of RC segments to represent a wire so that a tolerable runtime is achieved with sufficient accuracy.

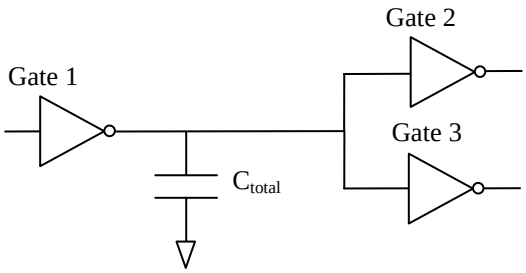



Fig. 1 Lumped capacitance model

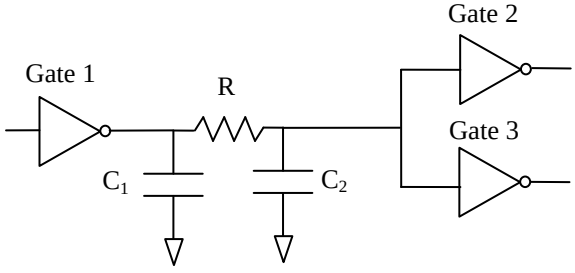



Fig. 2 Single π-model

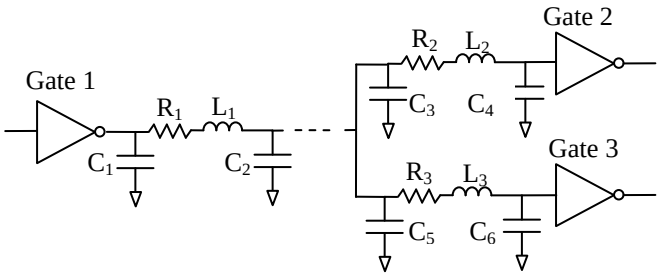



Fig. 3 Lumped RLC network model

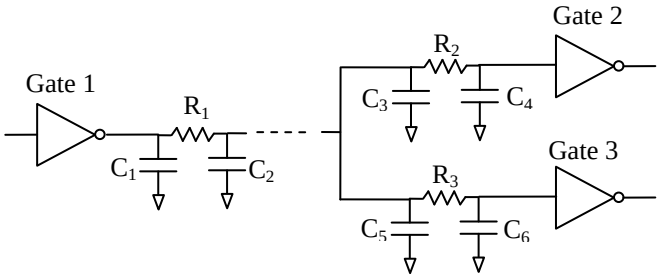



Fig. 4 Lumped RC network model

As illustrated in Fig. 5, STA requires an accurate computation of the delay from *A* to *B* based on the lumped RC network model, that is, the delay of a gate as well as the delay of interconnect it drives. A popular method to solve this problem is to decompose the problem into two steps [2], as illustrated in Fig. 6. In this approach, the whole RC network is replaced by an effective capacitance $C_{eff}$ attached directly to the gate output, which enables table look-up to compute the delay *d1* at point *A*'. Then the wire delay *d2* from *A*' to *B* is computed using an approximated ramp input acquired from the previous step.

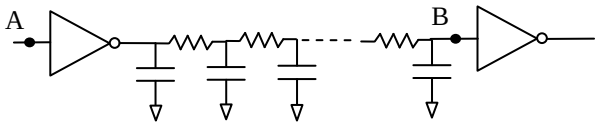



Fig. 5 Delay from A to B

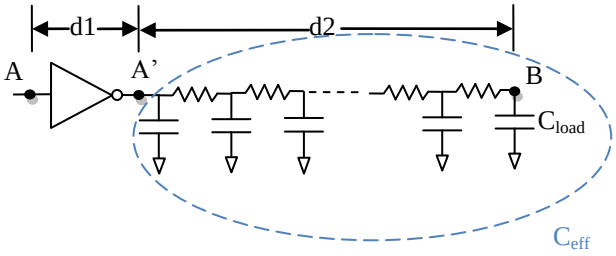



Fig. 6 Two-step approach

The *d1* computation usually comprises two steps: converting the RC network to a single π -model and then converting the single π -model to an effective capacitance, as shown in Fig. 7 [3-8, 15, 16]. O'Brien proposed a method in [3] to obtain the converted π -model parameters by matching the driving point admittance function of the actual gate load to the driving point admittance of the π -model up to 3rd order. $C_{eff}$ is converted from the single π -model based on the Thevenin gate model by matching the currents flowing into $C_{eff}$ and the actual load.

After the *d1* computation, the slew rate at *A*' and the delay from *A* to *A*' are obtained. The *d2* computation solves for the delay from *A*' to *B* based on a purely RC network, as shown in Fig. 8. Alpert demonstrates an accurate delay metric in [9] to compute the delay from the source of an RC network to any ending point. Agarwal gives slew rate metrics in [10] for computing the slew rate at any ending point of an RC network.

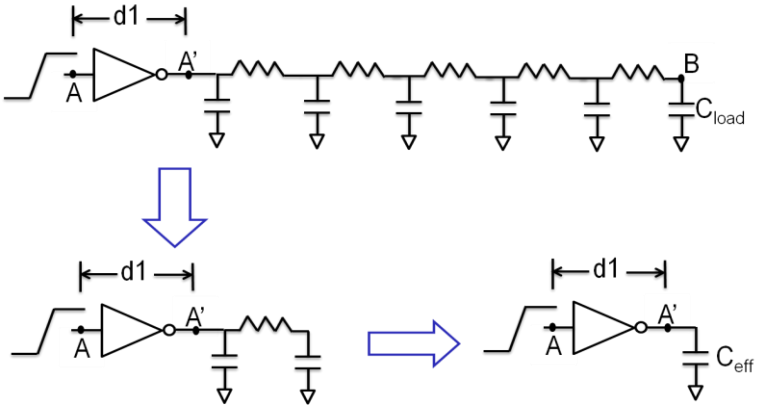



Fig. 7 d1 computation

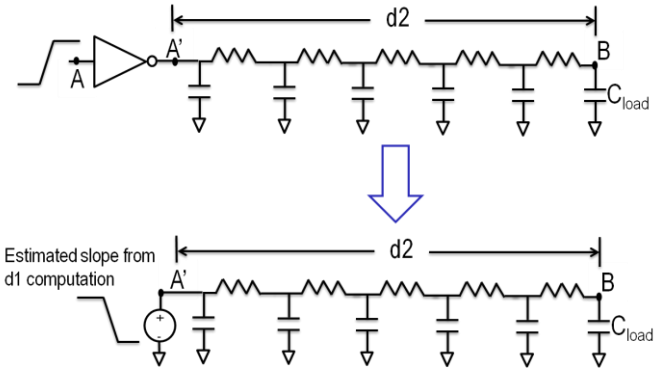



Fig. 8 d2 computation

The above two-step delay estimation method is popular and is used by the industry leading commercial static timing analysis tools. However we have found that the two-step approach is not accurate enough in many cases. Breaking the single delay calculation of Fig. 5 into the two-step approach of Fig. 6 doubles the error accumulation steps. First, converting a complex extracted net in contemporary technology to a single $\pi$ -model is substantially error-prone, especially when the wire delay tends to dominate the total delay in Fig. 5. Second, we have observed that the edge rate at node *A*' in Fig. 6 is not close to linear, which is what is used by the leading methods such as by Agarwal [10]. In fact, the simulated edge rates at *A*' in Fig. 6 for significant wire loads are observed to be best modeled as multi-slope piecewise linear fits, with a set of rather different slopes. Prior work has simply tried to model the input edge at *A*' with a single slope and we have found that this is a significant source of error.

We believe it would be best to entirely avoid having to deal with computing the waveform at *A*' and instead seek a gate plus wire delay model that captures in one step the entire delay from *A* to *B* in Fig. 5. We developed this new model and the algorithms for computing accurately the delay from *A* to *B* in Fig. 5, including the slew rate at *B*.

Our proposed algorithm accurately estimates the delay and slew rate at each wire endpoint based on arbitrary RC extractions, regardless of the number of RC sections. The new method avoids the error-prone two-step approach and instead computes *d1 + d2* in one step. Moreover, we developed a new algorithm to estimate the effective capacitance ($C_{eff}$) seen at the output of a gate. In addition, we developed an improved slew rate estimation metric for an endpoint such as *B*. Our delay and edge rate results compare favourably with respect to SPICE simulation results.

## II. EQUIVALENT DRIVER RESISTANCE AND $C_{EFF}$

Referring to Fig. 11, if an RC network is driven by a voltage source with a step input or single slope, there have been reports of effective algorithms for accurately estimating the delay and slope at endpoint *D*, as well as endpoints *A*, *B* and *C* [9]. In Fig. 11, $R_d$ can be viewed as the Thevenin equivalent resistance of the driver (voltage source).

Therefore, our key idea is to leverage this body of prior work for computing endpoint delays and slopes by finding an effective approach for computing the equivalent resistance of the driver or $R_d$. This is nontrivial since $R_d$ for a static CMOS gate is highly nonlinear and we wish to select one value for $R_d$ that yields accurate endpoint delays and slopes. Because of the nonlinearities, we find it necessary to find distinct $R_d$ values for each of the following for an endpoint: rise delay ($d_r$), fall delay ($d_f$), rise slope ($s_r$), and fall slope ($s_f$). If we are successful in finding such a set of four $R_d$'s, we can leverage and extend prior work to accurately compute endpoint delays and slopes in one step.

The time constant at *A* in Fig. 9 must match the time constant at *A* in Fig. 11. However, as the explicit resistance is not known for the gate in Fig. 9 and in addition the load consists of multiple π stages, it is not trivial to form this match. Instead, we iteratively compute a $C_{eff}$, as shown in Fig. 10, while seeking to match the time constant at the output of the gate in Fig. 10 with the time constant at point *A* in Fig. 11. The time constant at point *A* in Fig. 11 is given by: $\tau_A = R_d C_{total}$, where $C_{total} = C_1 + C_2 + C_3 + C_4$.

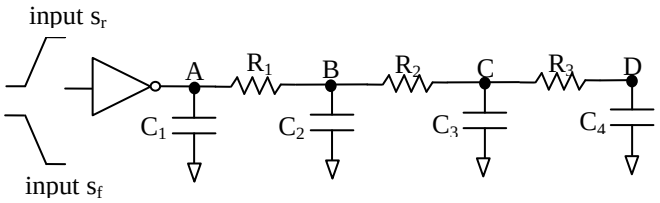



Fig. 9 A gate driving an extracted wire load

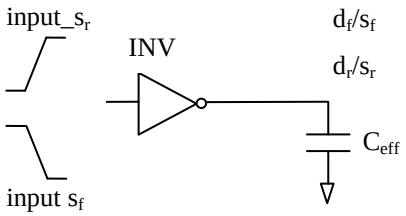



Fig. 10 Gate of Fig. 9 driving Ceff instead of the π-model, where the time constant at A in Fig. 9 must match the time constant at Ceff here. Shown are the rise delay ($d_r$), fall delay ($d_f$), rise slope ($s_r$), and fall slope ($s_f$)

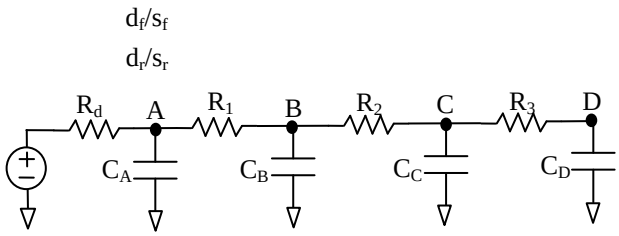
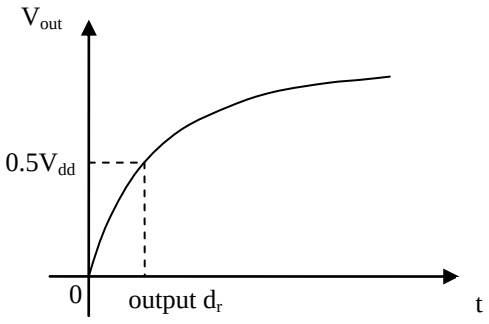



Fig. 11 Pure RC network with step function input, where the time constant at A in Fig. 9 must match the time constant at A in Fig. 11

The time constant (τ) at the output of the gate in Fig. 10 cannot be explicitly determined, but certainly is proportional to $C_{eff}$. Fig. 12 shows the charging and discharging curves for node *A* in Fig. 11.



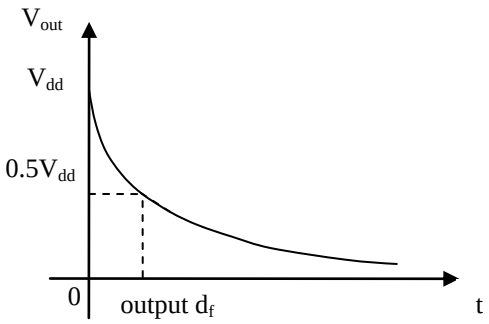



Fig. 12 Rising and falling delay match

Assuming an exponential charging waveform for node *A*, we expect $V_{out}$ to obey the following relationship, where $\tau_A = R_d C_{total}$:

$$V_{out} = V_{in}\left(1 - e^{-\frac{t}{\tau_A}}\right) \tag{1}$$

When $V_{out}$ reaches 50% of $V_{dd}$:

$$0.5V_{dd} = V_{dd}\left(1 - e^{-\frac{t}{\tau_A}}\right) \tag{2}$$

Rearranging:

$$e^{-\frac{output\ \ d_r}{R_{d\_rise\_delay}C_{total}}} = 0.5 \tag{3}$$

Now solving (3) for $R_d$:

$$R_{d\_rise\_delay} = \frac{output\ \ d_r}{-C_{total}\ln 0.5} = \frac{output\ \ d_r}{0.69315C_{total}} \tag{4}$$

The output rise delay in the numerator is identified from the .lib data for the load ($C_{eff}$) and Eq. (4) then yields the relevant $R_d$ for the output rising case. Note that the initial guess for $C_{eff}$ is $C_{total}$.

For delay matching when discharging the wire load (falling curve in Fig. 11):

$$V_{out} = V_{in}e^{-\frac{t}{\tau_A}} \tag{5}$$

When $V_{out}$ reaches 50% of $V_{dd}$:

$$0.5V_{dd} = V_{dd}e^{-\frac{t}{\tau_A}} \tag{6}$$

$$e^{-\frac{output\ \ d_f}{R_{d\_fall\_delay}C_{total}}} = 0.5 \tag{7}$$

Solving (3) for $R_d$ yields:

$$R_{d\_fall\_delay} = \frac{output\ \ d_f}{-C_{total}\ln 0.5} = \frac{output\ \ d_f}{0.69315C_{total}} \tag{8}$$

Likewise, the output fall delay in the numerator is identified from the .lib data for the load ($C_{eff}$) and Eq. (8) then yields the relevant $R_d$ for the output falling case. Again, the initial guess for $C_{eff}$ is $C_{total}$.

With respect to edge or slew rates, this is the time between $0.3V_{dd}$ and $0.7V_{dd}$ (or vice versa for falling edges). Fig. 12 marks the two time points where $0.3V_{dd}$ and $0.7V_{dd}$ are reached in the voltage transitions for node *A* in Fig. 11. We match the time duration between those two time points for the two circuits (Figs. 10 and 11).

For the charging waveform in Fig. 13, for $V_{out1}$ and $V_{out2}$:

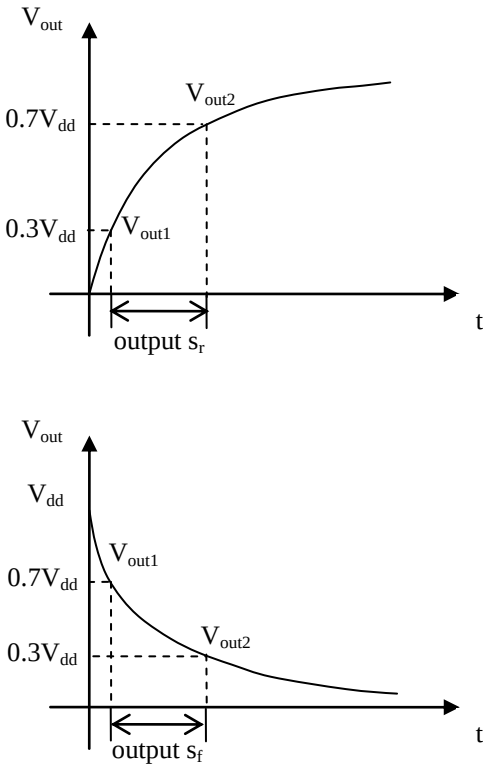



Fig. 13. Rising and falling slope match

$$\begin{aligned} 0.3V_{dd} &= V_{dd}\left(1-e^{-\frac{t_1}{\tau_A}}\right) \\ 0.7V_{dd} &= V_{dd}\left(1-e^{-\frac{t_2}{\tau_A}}\right) \end{aligned} \tag{9}$$

Solving for $t_1$ and $t_2$ yields:

$$\begin{aligned} t_1 &= -\tau_A \ln 0.7 \\ t_2 &= -\tau_A \ln 0.3 \end{aligned} \tag{10}$$

The duration from $t_1$ to $t_2$ is:

$$t_2 - t_1 = \tau_A(\ln 0.7 - \ln 0.3) \tag{11}$$

Since $\tau_A = R_d C_{total}$ and $t_2$ to $t_1$ is the slope, using (11) we obtain:

$$R_{d\_rise\_slope} = \frac{output \quad s_r}{0.8473C_{total}} \tag{12}$$

The result is the same for the discharging waveform slope:

$$R_{d\_fall\_slope} = \frac{output\ s_f}{0.8473 C_{total}} \tag{13}$$

For the previous two equations, the output rising and falling slopes are identified from the .lib data for the load (the initial guess for $C_{eff}$ is $C_{total}$.) and the equations then yield the relevant $R_d$ for the rising and falling slopes.

Given that we have computed $R_d$, albeit with an overly large initial estimate for $C_{eff}$ (equal to $C_{total}$) we now refine our approximation of $C_{eff}$ by using the $R_d$ just computed. We developed a closed-form expression for $C_{eff}$ for this purpose. Separate $C_{eff}$'s are calculated for each of rise delay, rise slope, fall delay and fall slope since there is a different $R_d$ for each case.

As mentioned earlier, we seek to match the time constant at the output of the gate in Fig. 10 (with load $C_{eff)}$ with the time constant at point $A$ in Fig. 11. That is, we require $\tau = \tau_A$, and therefore $\tau = \tau_A = R_d C_{total}$.

Consider a charging event at the endpoints ($A$, $B$, $C$ and $D$) in Fig. 11, starting from a completely discharged state. $C_{eff}$ is correctly identified when the total charge amount accumulated at endpoints $A$, $B$, $C$, $D$ equals the charge accumulated on the capacitor $C_{eff}$ in Fig. 10 when the voltage on $C_{eff}$ reaches the 50% point, occurring at time $t_1 = \tau \ln 2 = \tau_A \ln 2$.

The charge accumulated at each endpoint in Fig. 11 is illustrated in Fig. 14, where the sum of the $C_i * \Delta V_i$'s must equal the charge accumulated on the capacitor $C_{eff}$ in Fig. 10 at time $t_1$, and is governed by the following equations:

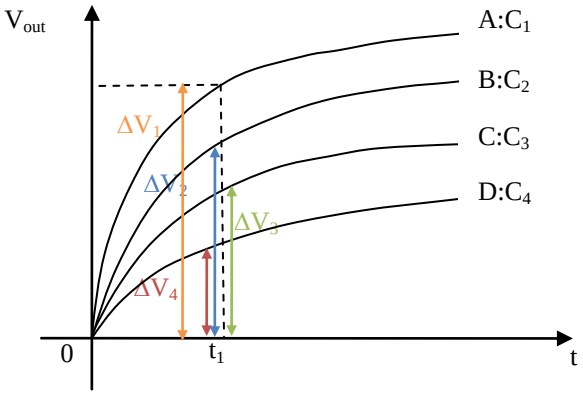



Fig. 14 Charging curve for each point in Fig. 11

$$\begin{aligned}
\Delta Q_A &= C_A \Delta V_A = C_A \left(1 - e^{-\frac{t_1}{\tau_A}}\right) V_{dd} \\
\Delta Q_B &= C_B \Delta V_B = C_B \left(1 - e^{-\frac{t_1}{\tau_B}}\right) V_{dd} \\
\Delta Q_C &= C_C \Delta V_C = C_C \left(1 - e^{-\frac{t_1}{\tau_C}}\right) V_{dd} \\
\Delta Q_D &= C_D \Delta V_D = C_D \left(1 - e^{-\frac{t_1}{\tau_D}}\right) V_{dd}
\end{aligned} \tag{14}$$

where:

$$\begin{aligned}
\tau_A &= R_d (C_A + C_B + C_C + C_D) \\
\tau_B &= R_d C_A + (R_d + R_1)(C_B + C_C + C_D) \\
\tau_C &= R_d C_A + (R_d + R_1) C_B + (R_d + R_1 + R_2)(C_C + C_D) \\
\tau_D &= R_d C_A + (R_d + R_1) C_B + (R_d + R_1 + R_2) C_C + (R_d + R_1 + R_2 + R_3) C_D
\end{aligned} \tag{15}$$

Also in Fig. 10:

$$\Delta Q_{eff} = C_{eff}\Delta V_{eff} = C_{eff}\left(1-e^{-\frac{t_1}{\tau_A}}\right)V_{dd} \tag{16}$$

where τ was replaced by $\tau_A$ since they are to be made equal. Note that the ratio of $t_1$ to τ in (16) is ln2. In order to match the charge accumulated:

$$\Delta Q_{(A+B+C+D)} = \Delta Q_{A'} \tag{17}$$

And thus:

$$C_A\left(1-e^{-\frac{t_1}{\tau_A}}\right)V_{dd} + C_B\left(1-e^{-\frac{t_1}{\tau_B}}\right)V_{dd} + C_C\left(1-e^{-\frac{t_1}{\tau_C}}\right)V_{dd} + C_D\left(1-e^{-\frac{t_1}{\tau_D}}\right)V_{dd} = C_{eff}\left(1-e^{-\frac{t_1}{\tau_A}}\right)V \tag{18}$$

where $t_1/\tau_A = \ln 2$.

The effective capacitance is then given by:

$$C_{eff} = C_A + \left(\frac{1-e^{-\frac{t_1}{\tau_B}}}{1-e^{-\ln 2}}\right)C_B + \left(\frac{1-e^{-\frac{t_1}{\tau_C}}}{1-e^{-\ln 2}}\right)C_C + \left(\frac{1-e^{-\frac{t_1}{\tau_D}}}{1-e^{-\ln 2}}\right)C_D \tag{19}$$

which simplifies to:

$$C_{eff} = C_A + 2\left[\left(1-e^{-\frac{t_1}{\tau_B}}\right)C_B + \left(1-e^{-\frac{t_1}{\tau_C}}\right)C_C + \left(1-e^{-\frac{t_1}{\tau_D}}\right)C_D\right] \tag{20}$$

For the discharging case in Fig. 15, the effective capacitance is derived similarly.

$$C_A V_{dd}\left(1-e^{-\frac{t_1}{\tau_A}}\right) + C_B V_{dd}\left(1-e^{-\frac{t_1}{\tau_B}}\right) + C_C V_{dd}\left(1-e^{-\frac{t_1}{\tau_C}}\right) + C_D V_{dd}\left(1-e^{-\frac{t_1}{\tau_D}}\right) = C_{eff} V_{dd}\left(1-e^{-\frac{t_1}{\tau_A}}\right) \tag{21}$$

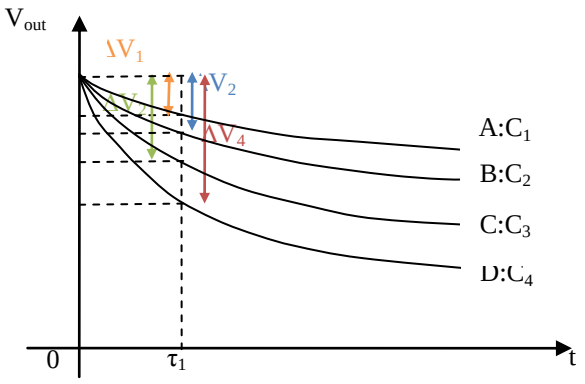



Fig. 15 Discharging curve for each point in Fig. 11

Solving Eq. (21) the effective capacitance is:

$$C_{eff} = C_A + \left(\frac{1-e^{-\frac{t_1}{\tau_B}}}{1-e^{-\ln 2}}\right)C_B + \left(\frac{1-e^{-\frac{t_1}{\tau_C}}}{1-e^{-\ln 2}}\right)C_C + \left(\frac{1-e^{-\frac{t_1}{\tau_D}}}{1-e^{-\ln 2}}\right)C_D \tag{22}$$

which simplifies to:

$$C_{eff} = C_A + 2\left[\left(1-e^{-\frac{t_1}{\tau_B}}\right)C_B + \left(1-e^{-\frac{t_1}{\tau_C}}\right)C_C + \left(1-e^{-\frac{t_1}{\tau_D}}\right)C_D\right] \quad (23)$$

Note that (20) and (23) are the same, and as expected, the effective capacitance is the same for both the charging and discharging cases.

Using this new $C_{eff}$ we again look to the *.lib* data to identify the new delay at the output of the gate for this load. That in turn allows us to compute a more accurate $R_d$ using (4), the formula for $R_d$ (in this situation, for the rising case), where the numerator is extracted from the .lib data. After obtaining the new $R_d$ value, (20) or (23) is once again used to extract an improved estimate for $C_{eff}$.

This process iterates until both $C_{eff}$ and $R_d$ converge. Upon convergence, the identified $C_{eff}$ when looked up in the *.lib* file for the proper slope yields the correct delay at the output of the gate. The corresponding $R_d$ yields exactly this $C_{eff}$. If one were to replace the gate with a resistor of value $R_d$ the delay at the near point of the wire would be matched with the delay at the output of the gate. We use this notion to declare that we have found the most effective $R_d$ to replace the gate in the netlist, arriving at Fig. 16.

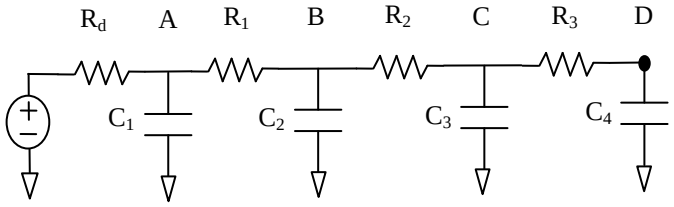



Fig. 16 Pure RC network with accurate $R_d$

The four algorithms we developed for respectively rise delay, fall delay, rise slop and fall slop are shown next.

*Algorithm: Compute $R_d$ for Rise Delay*

1. *For the worst-case input to output rising arc delay, use the input slope information to extract the rise delay ($d_r$) from the .lib data using $C_{total}$.*
2. *Insert $d_r$ found in Step 1 into Eq. (4) to find $R_d$.*
3. *Compute $C_{eff}$ using Eq. (20).*
4. *Using this $C_{eff}$, find $d_r$ from the .lib data and use Eq. (4) to find $R_d$.*
5. *Go to Step 3 until $R_d$ converges.*

*Algorithm: Compute $R_d$ for Fall_Delay*

1. *For the worst-case input to output falling arc delay, use the input slope information to extract the fall delay ($d_f$) from the .lib data using $C_{total}$.*
2. *Insert $d_f$ found in Step 1 into Eq. (8) to find $R_d$.*
3. *Compute $C_{eff}$ using Eq. (20).*
4. *Using this $C_{eff}$, find $d_f$ from the .lib data and use Eq. (8) to find $R_d$.*
5. *Go to Step 3 until $R_d$ converges*

*Algorithm: Compute $R_d$ for Rise_Slope*

1. *Given the gate input slope, extract the output rising slope information ($s_r$) from .lib data using $C_{total}$.*
2. *Insert $s_r$ found in Step 1 into Eq. (12) to find $R_d$.*
3. *Compute $C_{eff}$ using Eq. (20).*
4. *Using this $C_{eff}$, find $s_r$ from the .lib data and use Eq. (12) to find $R_d$.*
5. *Go to Step 3 until $R_d$ converges.*

*Algorithm: Compute $R_d$ for Fall_Slope*

1. *Given the gate input slope, extract the output falling slope information ($s_f$) from .lib data using $C_{total}$.*
2. *Insert $s_f$ found in Step 1 into Eq. (13) to find $R_d$.*

3. *Compute* $C_{eff}$ *using Eq. (20).*
4. *Using this* $C_{eff}$, *find* $s_f$ *from the .lib data and use Eq. (13) to find* $R_d$.
5. *Go to Step 3 until* $R_d$ *converges.*

## III. DELAY AND SLOPE CALCULATION

The next step is to determine the delays and slopes at the various endpoints of the wire, such as *A, B, C* and *D* in Fig. 16.

For the farthest endpoint of a wire, such as *D* in Fig. 16, the D2M metric method [12] is used to calculate the delay. The D2M delay metric equation is as follows:

$$D2M = \frac{m_1^2}{\sqrt{m_2}} \ln 2 \tag{24}$$

In Eq. (24) $m_1$ is the first moment of the impulse response for node *D* and $m_2$ is the second moment of the impulse response for node *D*. The *j*th moment for one node in the RC sections is defined as:

$$m_j = -\sum_{k=1}^{N} R_{ki} C_k m_{j-1}^{(k)} \tag{25}$$

Here $m_j^{(k)}$ is the *j*-th moment of the impulse response for the *k*-th node in the path (such as *A, B, C* or *D* in Fig. 16).

In Fig. 16, $m_1$ and $m_2$ for node *D* are computed according to (25) as:

$$m_1 = -\begin{bmatrix} R_d(C_1 + C_2 + C_3 + C_4) + R_1(C_2 + C_3 + C_4) \\ + R_2(C_3 + C_4) + R_3 C_4 \end{bmatrix} \tag{26}$$

$$m_2 = R_d C_1 m_1^1 + (R_d + R_1) C_2 m_1^2 + (R_d + R_1 + R_2) C_3 m_1^3 + (R_d + R_1 + R_2 + R_3) C_4 m_1^4 \tag{27}$$

In (26),

$$\begin{aligned} m_1^1 &= m_1 \\ m_1^2 &= R_d C_1 + (R_d + R_1)(C_2 + C_3 + C_4) \\ m_1^3 &= R_d C_1 + (R_d + R_1) C_2 + (R_d + R_1 + R_2)(C_2 + C_3) \\ m_1^4 &= R_d C_1 + (R_d + R_1) C_2 + (R_d + R_1 + R_2) C_3 + (R_d + R_1 + R_2 + R_3) C_4 \end{aligned} \tag{28}$$

Agarwal [10] provided slew rate metrics for computing the slew rate at any ending point of an RC network and complete the entire sentence, where a second-moment-based slope estimation empirical formula was proposed:

$$slope = r^{1/2} * \sqrt{2m_2 - m_1^2}\,(\ln 0.7 - \ln 0.3) \tag{29}$$

where *r* is an empirically derived adjusting factor:

$$r = -m_1 / \sqrt{m_2} \tag{30}$$

Our experiments on actual net extractions from industrial chips show that (28) is accurate for wire endpoints that are among the farthest away from the driver (source). However, for endpoints relatively near to the source, (29) can be substantially inaccurate. We have found worst-case errors for near endpoints to be in the vicinity of 20% compared to Hspice simulation.

Net topologies can assume a wide variety of shapes, so it is not obvious how to determine whether an endpoint is near-end (close to the source) or far-end (more towards the farthest endpoint from the source). In an attempt to quantify the relative nearness or farness from the source, we defined the notion of a *location factor*.

The location factor for an endpoint *x* is defined as the ratio of *x*'s first moment $m_1^x$ over the first moment of the endpoint (*max*) having the largest first moment (often but not always the endpoint farthest from the source) for this net.

$$location\_factor = \frac{m_1^x}{m_1^{\max}} \tag{31}$$

The smallest numerical value for the location factor we have seen for industrial circuits is approximately 0.6, so the range for the location factor is about [0.6, 1].

We have determined that (29) and (30) as proposed in [13, 14] work well for determining slopes for location factors in the upper half of this range, or [0.8, 1] and not well at all for the lower half of the range, or [0.6, 0.8]. As indicated in Table 1, we have found that much more accurate slopes are ascertained if the square root of the $r$ proposed in (30) is used for location factors in the range of [0.6, 0.8]. In Fig. 16 we show a RC network without any branches but actually our algorithm works for all scenarios including networks with various branches.

TABLE 1 ADJUSTING FACTORS ACCORDING TO LOCATION FACTORS

| **Location factor** | 0.6~0.8 | 0.8~1 |
|---|---|---|
| **Adjusting factor** | $r^{1/2}$ [sq. rt. of (27)] | $r$ [as in (27)] |

## IV. RUN TIME REDUCTION

We have noticed that for some gates in the layout the wire resistance is small enough that simply using the lumped capacitance model (*i.e.*, only using $C_{total}$) is sufficient. However, it is necessary to determine specific criteria as to when the lumped capacitance model is sufficient. For example, we require less than 1% error compared with Hspice simulation.

We introduced the notion of a "resistance limit" or $R_{limit}$ so that if the aggregate resistance of the net is less than $R_{limit}$, then the lumped capacitance model is sufficient to assure 1% accuracy with respect to Hspice simulation. We have observed that $R_{limit}$ not only depends on $C_{total}$ but also the type (logical function) and size of the driving gate. We therefore built a three-dimensional lookup table containing $R_{limit}$ values, depending on logical function (gate), size and $C_{total}$.

To fill the table, we selected a 10-section RC π-network to represent the wire load for each specific gate size. A 10-section network represented the smallest size for which there were diminishing returns from spending additional computation time. A range of $C_{total}$ values were selected, and these were evenly distributed over the ten π sections. The $C_{total}$ range selected is from approximately the smallest capacitive load seen on a chip to approximately the largest seen on a chip, typically a total of about 7 or 8 values.

For each capacitance value, using Hspice simulation, we find the largest total resistance (distributed evenly over the π sections) termed the $R_{limit}$ for which the difference in delay or slope between the lumped capacitance model and the π-section model is no more than 1%. Figs. 17 and 18 illustrate how $R_{limit}$ varies versus $C_{total}$ for two inverter drive strengths, inv_1 and inv_8, from an industrial 45 nm library.

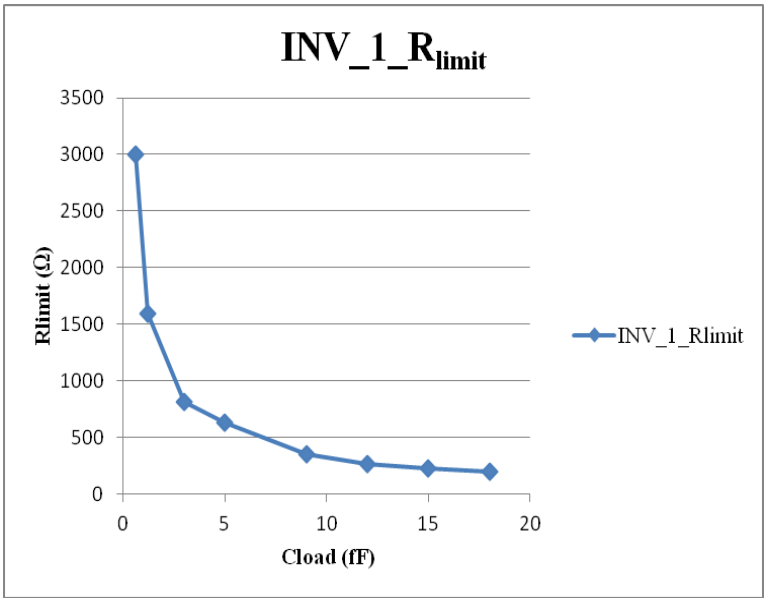



Fig. 17 $R_{limit}$ versus $C_{total}$ for INV_1

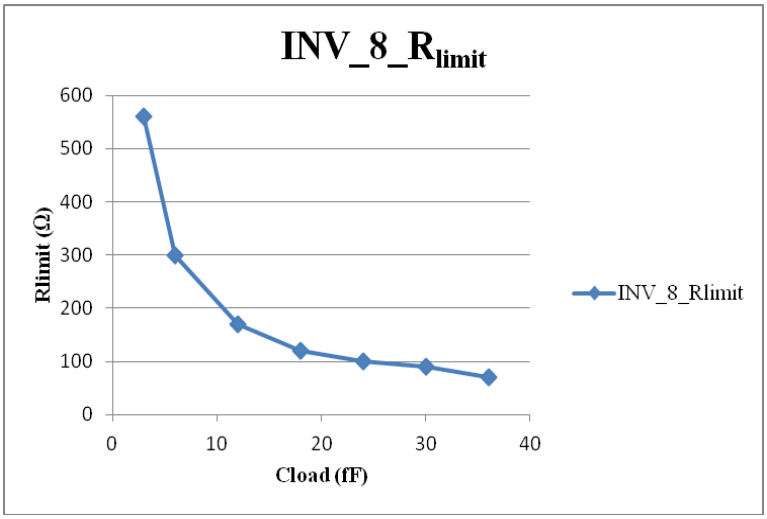



Fig. 18 $R_{limit}$ versus $C_{total}$ for INV_8

We generate tabular data of the type shown in Figs. 17 and 18 for each drive strength for each cell in the library. It is then straightforward to interpolate for any specific $C_{total}$ value.

Our procedure is as follows: Given a wire load for a specific gate, we sum the capacitances to obtain $C_{total}$. We use the $R_{limit}$ table (as in Figs. 17 and 18) to get $R_{limit}$ for this net, using interpolation or extrapolation as necessary. If the aggregate wire resistance for this net exceeds $R_{limit}$, then we use our new one-step approach (using the four algorithms in Section II for the equivalent driver resistances ($R_d$'s) as well as the equations in Section III) to obtain the delay and slope information. Otherwise, we simply use the lumped capacitance model for obtaining delay and slope for this net.

For an industrial chip consisting of over 100,000 standard cells (and nets) and having a total of 513,223 wire endpoints, a total of 50,233 nets including 231,070 sinks had aggregate resistances, for the specified $C_{total}$, which fell below their respective $R_{limit}$ and were thus eligible for the simpler lumped capacitance analysis. Table 2 shows the results for these 50,233 nets. Note that the worst-case error can slightly exceed 1% due to interpolation and also since real net extractions do not consist of 10 π-sections with uniform R and C distributions.

TABLE 2 SIMULATION RESULTS BASED ON LUMPED C MODEL

| | **Mean Error vs. Hspice** | **% endpoints with less than 1% or 1ps error vs. Hspice** | **Worst error vs. Hspice** |
|---|---|---|---|
| **Delay** | 0.85% | 98.2% | 1.2% |
| **Slope** | 0.91% | 97.6% | 1.5% |

Our delay and slope estimation algorithms are summarized here.

*Algorithm: Compute Endpoint Delays for Gate (g)*

1. *Note the logic function of* g *and its size, also compute* $R_{total}$ *and* $C_{total}$ *for the extracted net driven by* g.
2. Compute $R_{limit}$ based on information gathered in Step 1.
3. If $R_{total}$ is less than $R_{limit}$, use the lumped capacitance model to determine the endpoint delays.
4. Otherwise use *Algorithm Compute* $R_d$ *for Rise Delay* or *Algorithm Compute* $R_d$ *for Fall Delay* from Section II (depending on whether rise or fall delay, respectively, is desired for *g*) to obtain $R_d$.
5. *For any desired endpoint of this extracted net, use (24) to estimate the delay (rise or fall).*

*Algorithm: Compute Endpoint Slopes for Gate (g)*

1. *Note the logic function of* g *and its size, also compute* $R_{total}$ *and* $C_{total}$ *for the extracted net driven by* g.
2. Compute $R_{limit}$ based on information gathered in Step 1.
3. If $R_{total}$ is less than $R_{limit}$, use the lumped capacitance model to determine the endpoint slopes.
4. Otherwise use *Algorithm Compute* $R_d$ *for Rise Slope* or *Algorithm Compute* $R_d$ *for Fall Slope* from Section II (depending on whether rise or fall slope, respectively, is desired for *g*) to obtain $R_d$.

*5. For any desired endpoint of this extracted net, use (29), (30) and Table 1 to estimate the slope (rise or fall).*

The $R_d$ convergence requirement is set as 0.1% in our program. In such a criteria $R_d$ is converged within three iterations for all nets. And the average CPU time is 0.08 ms. The algorithm complexity is $O$(n).

## V. DELAY AND EDGE RATE RESULTS

We selected an industrial chip consisting of over 100,000 standard cells (and nets) and having a total of 513,223 wire endpoints to evaluate our new approach, which was implemented in C. In Table 3 we compare our results for each of the 513k endpoints versus those obtained by Hspice simulation, which yields the precise results. We also show the leading commercial STA tool's results versus Hspice.

TABLE 3 SIMULATION RESULTS BASED ON LUMPED C MODEL

| | **Mean error vs. Hspice** | **% endpoints with error: <1% or 1ps vs. Hspice** | **Max. error vs. Hspice** |
|---|---|---|---|
| **Delay (ours)** | 0.49% | 93.6% | 9.4% |
| **Delay (PrimeTime)** | 1.35% | 42.2% | 11.3% |
| **Slope (ours)** | 1.88% | 39.1% | 9.8% |
| **Slope (PrimeTime)** | 2.51% | 33.6% | 10.6% |

The mean errors for our results versus Hspice are shown in the first column of Table 3. The second column shows the percentage of the endpoints whose error was less than one percent or less than one picosecond, either of which could be considered as sufficiently accurate. The third column indicates the maximum error found among the 513k endpoints.

Note that our new algorithm substantially outperforms the leading commercial STA tool for delay, and gets about 94% of the sinks to within very good accuracy. Our algorithm also outperforms the commercial STA tool for edge rate determination. The worst-case slope error for our algorithm is 9.8% whereas we found that using previous work [13, 14], the worst-case error was 20%.

## VI. CONCLUSIONS

We have proposed new algorithms to accurately estimate the delay and slew rate at wire endpoints of an extracted RC network. The algorithm calculates an equivalent resistance to replace the driving gate of the extracted net so that the delay and slew rate at each endpoint can be estimated using established techniques (or metrics). Furthermore, we appreciably improved these delay and slope metrics. We also developed an approach that quickly determines whether or not the lumped capacitance model is sufficient for determining the delays and slopes for the endpoints of an extracted RC network. The delays and slopes produced by our new estimation algorithms are quite accurate with respect to Hspice simulation results, while substantially outperforming previously published work as well as the leading commercial tool (Synopsys' PrimeTime). Finally, the newly developed algorithm also estimates the effective capacitance ($C_{eff}$) seen by a gate. The future work includes improving the algorithm for providing a more accurate modelling of slope estimation and proving the proposed effective capacitance ($C_{eff}$) estimation methodology in this paper is a better one compared to commercial tools.

## REFERENCES

[1] H. Kaeslin, "Digital Integrated Circuit Design", Cambridge University Press, 2008, pp. 612-613.

[2] M. Celik, L. Pileggi and A. Odabasioglu, "IC Interconnect Analysis", Kluwer Academic Publishers, 2002, pp. 276-285.

[3] A. Kahng and S. Muddu, "Gate Load Delay Computation Using Analytical Models", *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems*, 1996, pp. 433-436.

[4] P. O'Brien and T. L. Savarino, "Modeling the driving-point characteristic of resistive interconnect for accurate delay estimation," *International Conference on CAD*, 1989, pp. 512-515.

[5] F. Dartu, N. Menezes, J. Qian and L. T. Pillage, "A gate-delay model for high speed CMOS circuits," *Proc. Design Automation Conference*, 1994, pp. 576-580.

[6] F. Dartu, N. Menezes, and L. T. Pileggi, "Performance computation for precharacterized CMOS gates with RC loads," *IEEE Trans. on Computer-Aided Design*, vol. 15, iss. 5, 1996, pp. 544-553.

[7] J. Qian, S. Pullela, and L. Pillage, "Modeling the 'Effective Capacitance' for the RC interconnect of CMOS gates," *IEEE Transaction on CAD of Integrated Circuits and Systems*, vol. 13, iss. 12, 1994, pp. 1526-1535.

[8] S. Fang, Z. Huang, A. Kurokawa, and Y. Inoue, "Calculating the effective capacitance for interconnect loads based on Thevenin Model," *International Conference on Communications, Circuits and Systems*, 2006, pp. 2474-2477.

[9] S. Fang, Z. Huang, A. Kurokawa, and Y. Inoue, "An advanced model for calculating the effective capacitance considering input waveform effect," *International Conference on Communications, Circuits and Systems*, 2008, pp. 1088-1092.

[10] K. Agarwal, D. Sylvester and D. Blaauw, "An effective capacitance based driver output model for on-chip RLC interconnects", *Design Automation Conference*, 2003, pp. 376-381.

[11] K. Agarwal, D. Sylvester and D. Blaauw, "A library compatible driver output model for on-chip RLC transmission lines", *IEEE Transactions on CAD of Integrated Circuits and Systems*, 2004, pp. 128-136.

[12] C. Alpert, A. Devgan, and C. V. Kashyap, "RC delay metrics for performance optimization," I*EEE Transaction on CAD of Integrated Circuits and Systems*, vol. 20, iss. 5, 2001, pp. 571-582.

[13] K. Agarwal, D. Sylvester and D. Blaauw, "Simple metrics for slew rate of RC circuits based on two circuits moments", *Proc. Design Automation Conference*, 2003, pp. 950-953.

[14] K. Agarwal, D. Sylvester and D. Blaauw, "A simple metric for slew rate of RC circuits based on two circuit moments", *IEEE Transactions on CAD of Integrated Circuits and Systems*, 2004, pp. 1346-1354.

[15] M. Jiang, Z. Huang, A. Kurokawa, S. Fang and Y. Inoue, "Accurate method for calculating the effective capacitance with RC loads based on the Thevenin model", *IEICE Trans. Fundamentals*, 2009, pp. 2531-2539.

[16] M. Jiang, Q. Li, Z. Huang and Y. Inoue, "A non-iterative effective capacitance model for CMOS gate delay computing", *ICCCAS*, 2010, pp. 896-900.

**Zhao Wang** acquired his Ph.D. in Electrical Engineering from The University of Texas at Dallas, TX in 2012. Before that he obtained his master degree in Electrical Engineering from The University of Texas at Dallas, TX in 2009. His main research areas when pursuing Ph.D. degree were circuit optimization and CAD algorithm development.

He is currently working at Apple Inc., Cupertino, CA as circuit designer. His current research interests are VLSI design, CAD algorithm development and new emerging technology deployment.

**Carl M. Sechen** acquired his Ph.D. in Electrical Engineering from UC Berkeley, CA in 1987. Before that He obtained his master degree in Electrical Engineering from MIT, MA in 1977 and bachelor degree in Electrical Engineering from University of Minnesota in 1975.

He is currently a professor at The University of Texas at Dallas, TX. His research areas include design and computer-aided design of integrated circuits. Before joining UT Dallas he had been with University of Washington and Yale University.

Dr. Sechen was rewarded as IEEE fellow in 2002.