修士論文

微細化されたLSIにおける
ツリー構造配線への最適リピータ挿入

1999年2月5日提出

指導教官
桜井貴康教授

東京大学大学院工学系研究科
電子工学専攻
学籍番号76444

井高　康仁

第1章　　序論

近年、集積回路技術がサブミクロン領域に入るにつれて、トランジスタ数は数百万から数千万トランジスタに達している。トランジスタ数の増大につれて、それらを結ぶ配線の数も膨大な量に膨れ上がっている。チップ面積が配線面積に制限されないように配線の幅、厚さ、間隔もトランジスタのスケーリングと共に可能な限りスケーリングしなくてはならない。それに加えて、多くの配線領域を得るために多層配線技術を用いて設計されるため、ますます LSI 中の配線の占める部分は増大していくと考えられる。

スケーリングとともに配線部分の抵抗が増大するため、配線遅延の増加が問題となってきている。配線抵抗と容量は長さに比例するため、それらの積で決まる配線遅延の成分は長さに対し2乗で増大する。チップサイズの増大と共に配線長の最大値は増加するため、この問題は深刻である。さらにゲート遅延の減少を考えると、ゲート遅延に対する配線遅延のますます増大すると考えられる。そのため、配線遅延が小さくなるように設計することが高速 LSI の設計にとって必要不可欠となる。

図1．1 に SIA のロードマップ[1]を示す。横軸にゲート幅、縦軸にゲート遅延や配線遅延をとってある。銅配線や低誘電体材料を使って、抵抗、容量の増加を抑える場合であっても、配線遅延は増加し回路の遅延を制限するため、今後は長い配線を減らしたり、これから述べるリピータ挿入によって配線遅延の増加を緩和することが必要となってくる。

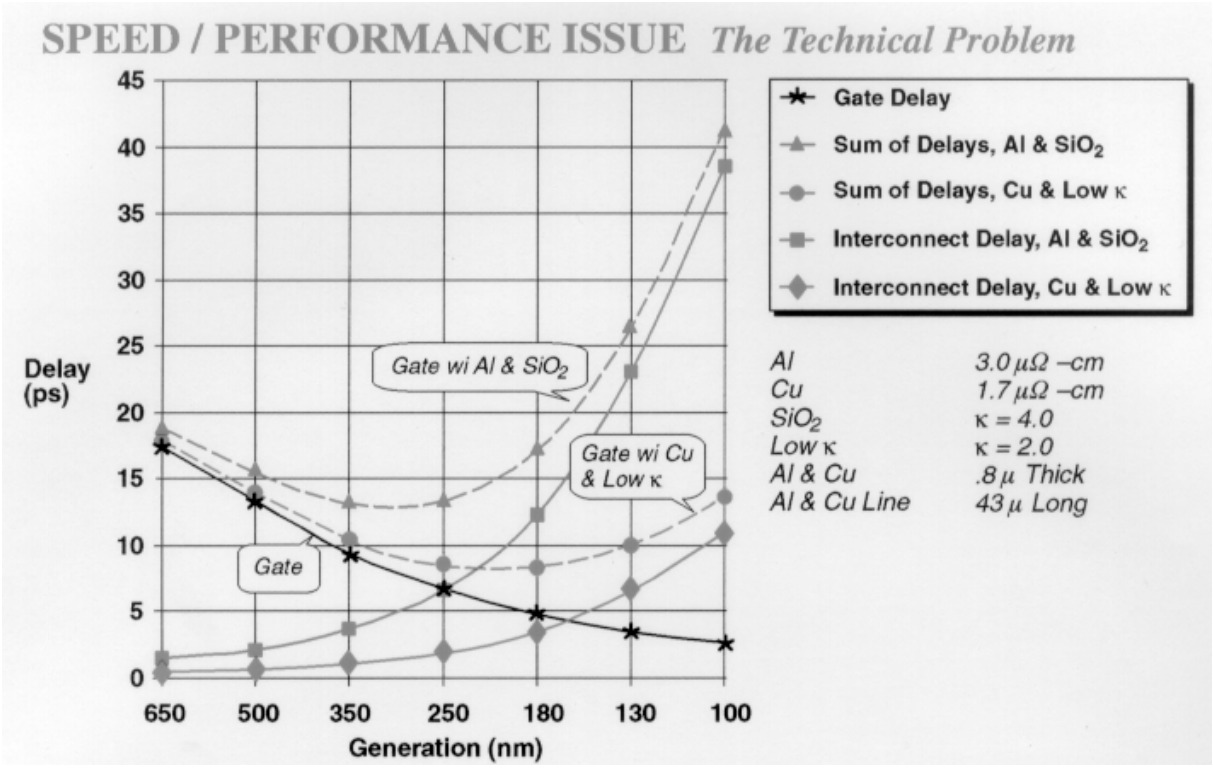

図 1.1 SIA roadmap

1.1. Repeater

配線遅延の増加を緩和する方法として、リピータ挿入の方法[2],[3]がある。配線の抵抗と容量の積である配線遅延が2乗で増加するため、配線を分割することにより増加を線形程度の増加に減少させることが目的である。

図 1.2 のようにモデル化された問題を考える。入力から出力まですべてのトランジスタが同じサイズで、等間隔であると仮定し、全体の配線抵抗をR_{line}、配線容量C_{iline}とし、最小サイズのリピータの入力容量をC_0、出力抵抗をR_0で近似することにする。

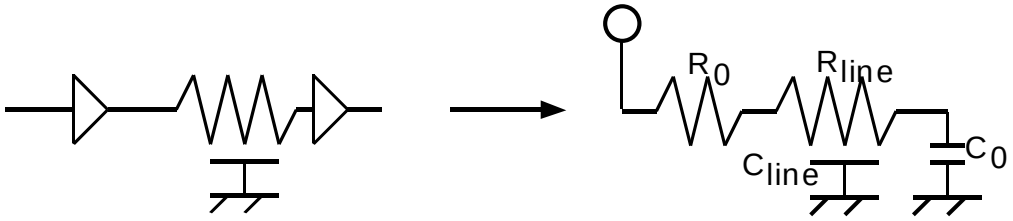

図1．2 トランジスタのモデル化

Elmore delay（2.1節参照）は次のようになる。

$$\mathrm{Delay} = R_0\left(C_{\mathrm{line}} + C_0\right) + R_{\mathrm{line}}\left(0.5C_{\mathrm{line}} + C_0\right) \tag{1.1}$$

トランジスタのチャネル幅が最小サイズのh倍であり、それをk段挿入したときの式は、リピータ容量がサイズに比例し、リピータ等価抵抗がサイズに反比例するとすると、遅延時間は次の式で表される。

$$\mathrm{Delay} = k\left(\frac{R_0}{h}\left(\frac{C_{\mathrm{line}}}{k} + hC_0\right) + \frac{R_{\mathrm{line}}}{k}\left(0.5\frac{C_{\mathrm{line}}}{k} + hC_0\right)\right) \tag{1.2}$$

と表される。ここでこれをkについて微分して最小の遅延を与えるkを求めると

$$k = \sqrt{\frac{R_{\mathrm{line}}C_{\mathrm{line}}}{2R_0C_0}} \tag{1.3}$$

であり、また同様にhについての最適値は

$$h = \sqrt{\frac{R_0C_{\mathrm{line}}}{R_{\mathrm{line}}C_0}} \tag{1.4}$$

である。

このようにして単一配線での最適リピータサイズ、および最適リピータ数が求められる。

1.2. Cascaded driver

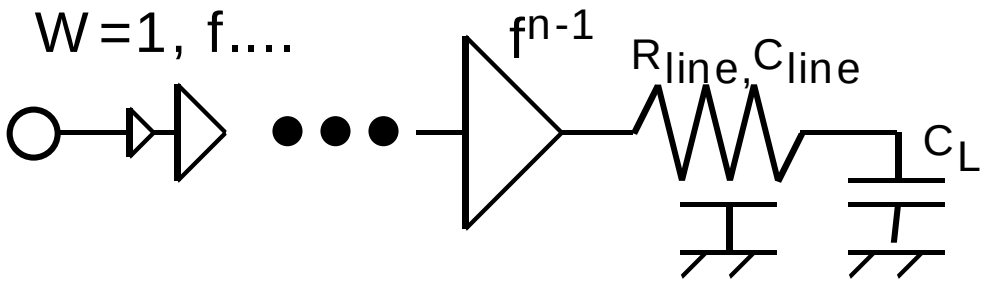

図1．3　Cascaded driver

大きな容量性の負荷である場合、最適リピータサイズが大きいため、段々とサイズを増して、最終

段のリピータが負荷を駆動するのに十分な大きさをもつようにリピータを連鎖させる cascaded driver の方法が有効である。。図1．3 のように n 段のリピータで前後の大きさの比を f とすると遅延時間の和は次のようになる。

$$\begin{aligned} \mathrm{Delay} &= (n-1) f R_0 C_0 \\ &+ \left(\frac{R_0}{f^{n-1}} + 0.5 R_{int} \right) C_{int} + \left(\frac{R_0}{f^{n-1}} + R_{int} \right) C_L \end{aligned} \tag{1.5}$$

と f や n で微分することにより最適値は

$$\begin{aligned} f &= e \\ n &= \ln\left(\frac{C_{int} + C_L}{C_0} \right) \end{aligned} \tag{1.6}$$

となる。すなわち buffer の大きさを自然対数 e 倍だけ大きくした buffer を挿入していけばよいことがわかる。

1.1、1.2 で示したよう単一の長い配線を駆動する場合、最初に cascaded driver によって、駆動力を増し、その後最適サイズのリピータを挿入することにより、高速化することができる。

1.3. リピータ挿入の効果

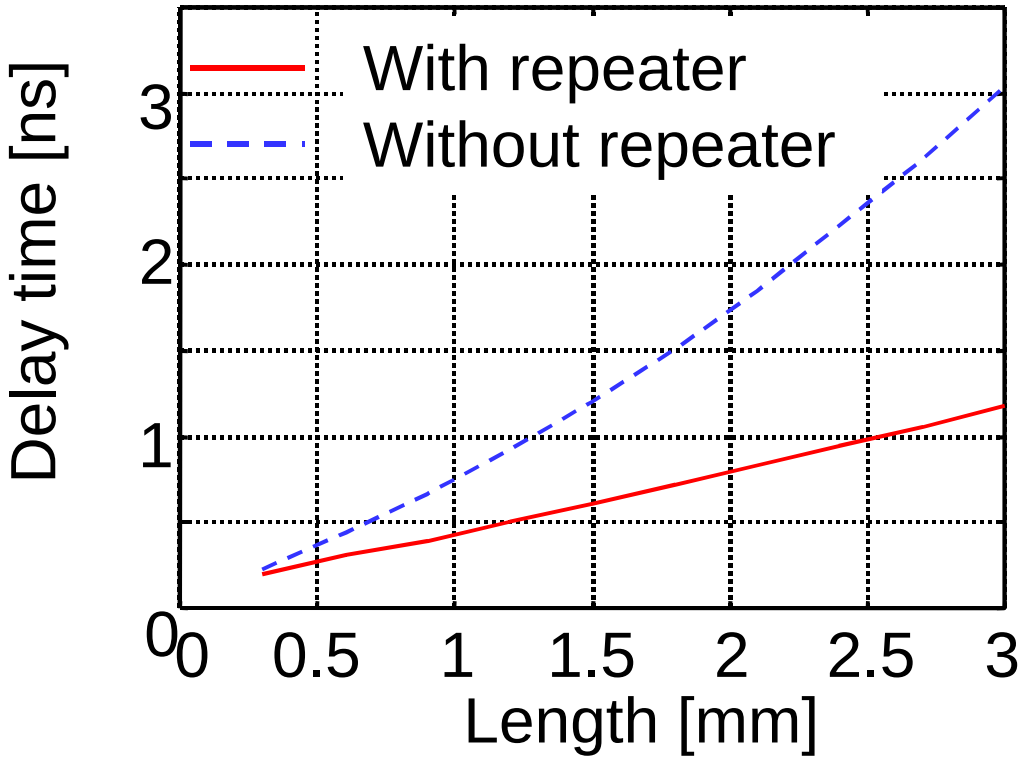

図 1.5　リピータ挿入の効果

図 1.5 はリピータ挿入の効果を表すグラフである。配線長の増加とともにリピータ挿入の効果があらわれることがわかる。図は配線幅 0.1μm の例であり、トランジスタのゲート幅は 0.5μm のパラメータを用いているため、実際ゲート幅 0.1μm となった場合はリピータ容量が減るため最適リピータ数、最適リピータサイズ共に増加し、リピータ挿入の効果は顕著にあらわれる。

1.4. 本研究の目的

上記の方法では Elmore delay より最適値を求めたが、実際のトランジスタや分布定数線路で遅延

時間を考えた場合、Elmore のモデルとの誤差があらわれてくる。そのため、分布定数線路やリピータの精度のよい近似式を考えなくてはならない。

また、配線遅延を減少させるためのリピータ挿入の方法として一般的な repeater 及び cascaded driver を上に記したが、実際の配線に適用する場合、配線は分岐したツリー構造で考えるのが一般的である。そのため、分岐点でのリピータ挿入の方法を考えなくてはならない。また、出力端の負荷の変化にも対応させることで効率を増すことができる。また、遅延遅延の最適値にリピータ挿入を行った場合、リピータでの消費電力成分が増加し、配線での消費電力の 1.6 倍(4 章より)となる。そのため低消費電力設計を考える場合、配線での消費電力を抑えた形でのリピータ挿入を考える必要がある。

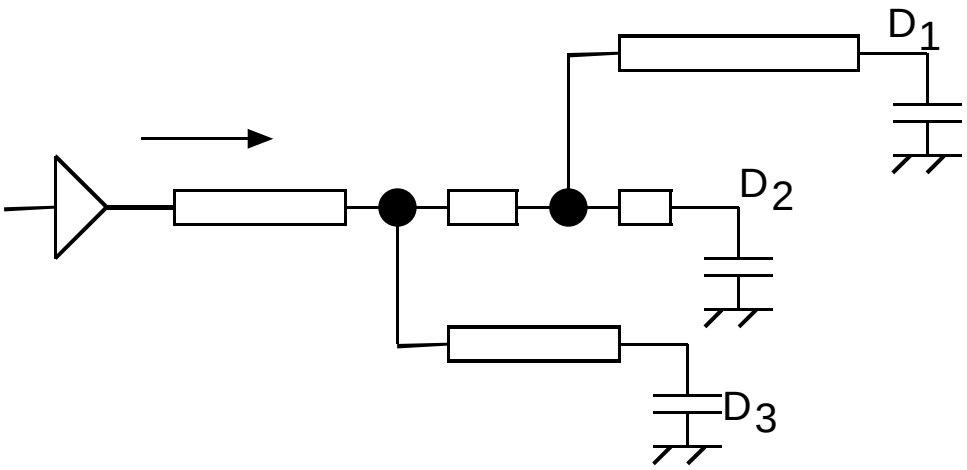

図 1.6　ツリー構造回路

今回、配線遅延解析の方法として 2 章で moment matching 法を用いた本研究での配線遅延モデルを示し、3 章ではツリー構造配線における遅延時間最適化のための方法を考察した。4 章では、低消費電力設計に向けて遅延時間・消費電力積最適化の方法を示し、消費電力と最適リピータサイズ、最適リピータ数の関係を示す。5 章で結論を述べ、6 章で今後の課題を示す。

第2章　　遅延モデル式

2.1. Elmore delay

MOS集積回路のタイミング解析を行うために、トランジスタの出力と配線を抵抗と容量でツリー型で表現する方法が使われる。ここで扱われるツリーはすべてのノードからグラウンドへの容量があり、浮いたノード（つまり抵抗によって接続されていないノード）がなく、グラウンドに直接つながっている抵抗や抵抗によるループも存在しない。MOS集積回路の遅延は一般にこのようなRCツリーモデルを解析することにより扱われる。現在もっとも一般的な方法は Elmore の方法である。

ノード数NのRCツリー回路において、ノード i 上での Elmore delay は、

$$T_{Di} = \sum_{k=1}^{N} R_{k\,i} C_k \qquad (2.1)$$

で定義される。ここで R_{ki} は入力ノードとノード k との（唯一の）経路と、入力ノードとノード i との（唯一の）経路との共有する部分上の抵抗であり、C_k はノード k とグランドとの間の容量である。すなわち図 2.1 のような回路においては、Elmore delay は、

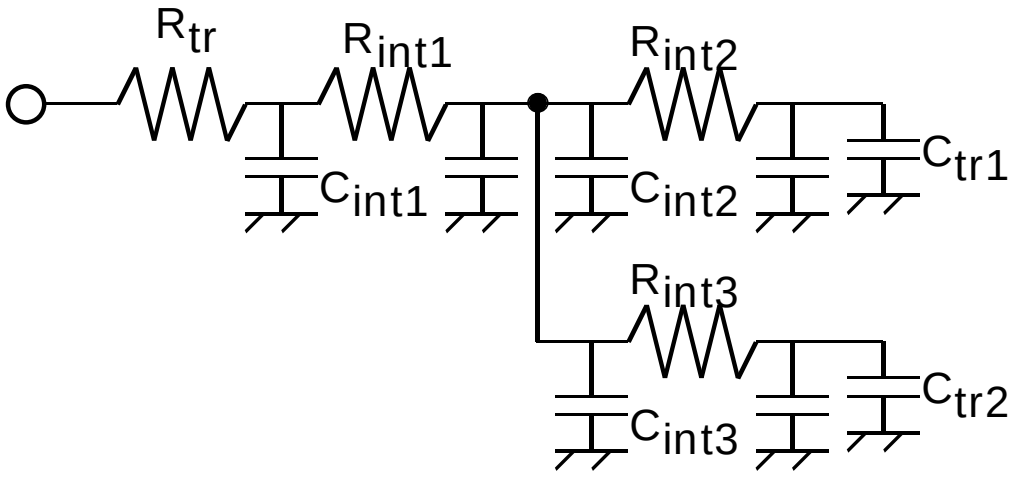

図 2.1 Elmore delay

$$\begin{aligned}\tau_1 &= R_{tr}\left(C_{int1}+C_{int2}+C_{int3}+C_{tr1}+C_{tr2}\right)\\ &+R_{int1}\left(\frac{1}{2}C_{int1}+C_{int2}+C_{int3}+C_{tr1}+C_{tr2}\right)\\ &+R_{int2}\left(\frac{1}{2}C_{int1}+C_{tr1}\right)\end{aligned} \qquad (2.2)$$

と表される。また、今回全ての配線を分布定数線路で扱ったが、分布定数線路での Elmore delay は図 2.1 に示したようなπ型回路で扱うことができる。

2.2. Moment matching 法

本論文では LSI 中の線路を分布定数線路として考えることにする。まず、よく知られている一本の分布定数線路の式から出発してこれをツリー構造の回路での遅延時間の計算に適用することを考えることにする[5]。

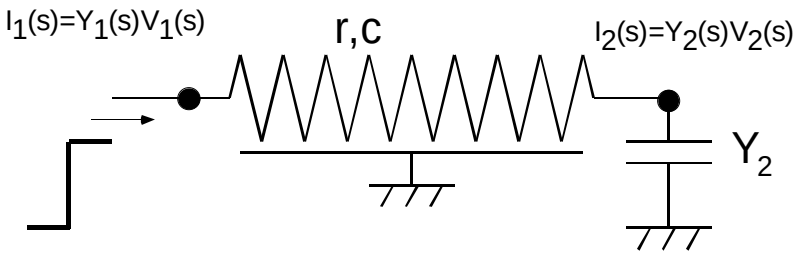

図 2.2　分布定数線路

図 2.2 の様な一本の分布定数線路では Laplace 変換を行ったあとの s 領域での電圧、電流の関係は次のように表されることが知られている。

$$\begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} = \begin{pmatrix} \cosh\gamma & -\frac{r}{\gamma}\sinh\gamma \\ -\frac{r}{\gamma}\sinh\gamma & \cosh\gamma \end{pmatrix} \begin{pmatrix} V_2(s) \\ I_2(s) \end{pmatrix} \tag{2.3}$$

ただし、$\gamma = \sqrt{rcs}$

$I_2 = Y_2 V_2(s)$、$I_1 = Y_1 V_1(s)$と各点でのアドミタンスを定義すると次の様な関係となる。

$$Y_1 = \frac{\frac{\gamma}{r}\sinh\gamma + Y_2\cosh\gamma}{\cosh\gamma + \frac{r}{\gamma}Y_2\sinh\gamma} \tag{2.4}$$

これをもとにツリー構造の回路を考える。図 2.3 の回路では、枝分かれ部分での電流値は電流保存則より求めることができ、図 2.3 の様に各点で電圧、電流、アドミタンスを定義すると、

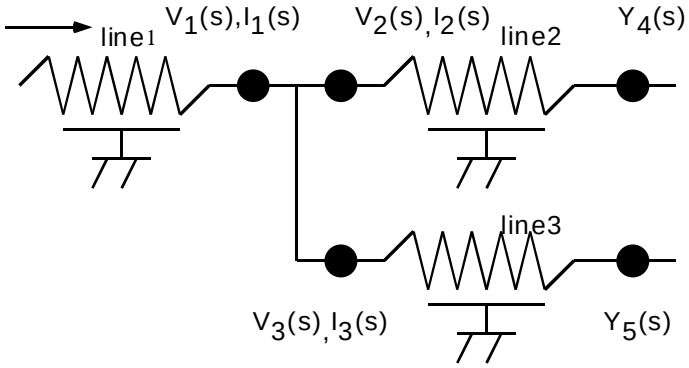

図 2.3　分岐部分での計算法

$$I_1 = Y_2 V_2 + Y_3 V_3$$
$$= V_1 \left(\frac{\frac{\gamma_2}{r_2}\sinh\gamma_2 + Y_4\cosh\gamma_2}{\cosh\gamma_2 + \frac{r_2}{\gamma_2}Y_4\sinh\gamma_2} + \frac{\frac{\gamma_3}{r_3}\sinh\gamma_3 + Y_5\cosh\gamma_3}{\cosh\gamma_3 + \frac{r_3}{\gamma_3}Y_5\sinh\gamma_3} \right) \qquad (2.5)$$

となり、枝分かれの結合部分でのアドミタンスを求めることができる。このように式の演算により求められる成分の和を求めることにより枝分かれ部分でのアドミタンス成分を計算できることがわかる。

また、アドミタンスは s＝0の周りで Taylor 展開したもので計算することにすると、式(2.5)の演算においては、高次の項の成分が低次の項に全く影響しない。そのため必要な精度までの次数を計算することができそのため必要に応じて計算量を軽減することができる。

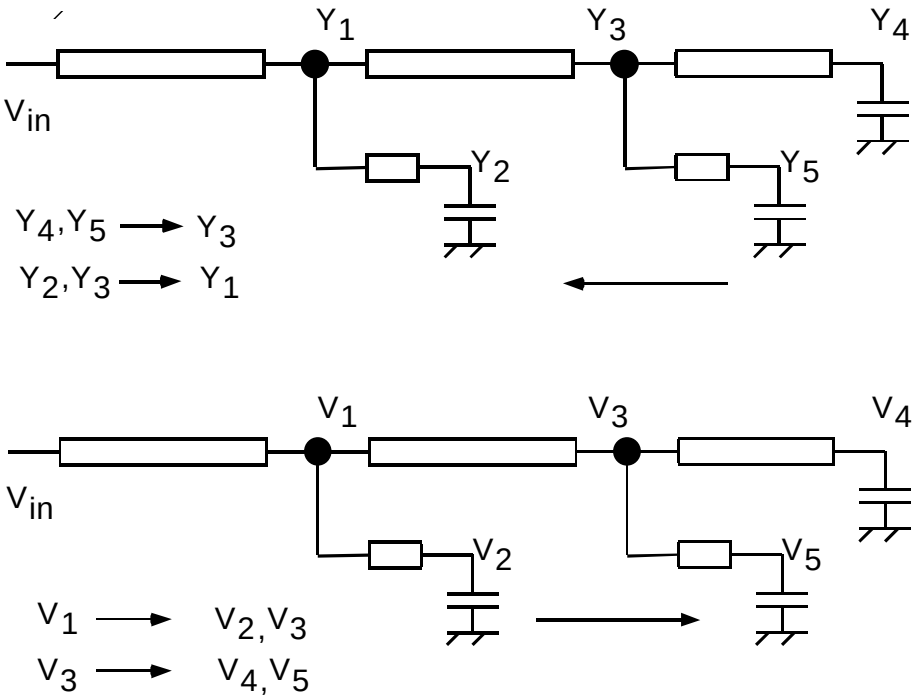

図 2.4　ツリー構造配線のアドミタンス、電圧計算法

以上示した方法を応用することにより、ツリー構造の回路について各ノードの電圧値をs領域で計算することができるようになることがわかる。すなわち、図 2.4 のような回路では、まず出力ノードのアドミタンスY_4、Y_5から、枝分かれの合流部分のアドミタンスY_3を計算できる。次に同様にY_3、Y_2からY_1を計算する。入力端電圧V_{in}がわかっていれば、逆にそれぞれの出力端に向かって計算することにより、s領域での V_1 、V_2、V_3 、V_4 、V_5を求めることができる。

前節の方法によってs領域での分布定数線路のツリー構造回路でのs領域での式は正確に求めることができる。問題はこれをt領域（時間領域）に変換することであるが、この変換は以下の moment matching 法をを用いる。
出力電圧を例えば以下の式の様に仮定する。

$$v(t)=\sum_{j=1}^{q} k_j e^{p_j t} \tag{2.5}$$

この式に Laplace 変換を行ったあと、s＝0の周りで展開することにより以下のようになる。

$$\begin{aligned} V(s) &= \sum_{j=1}^{q} \frac{k_j}{s-p_j} \\ &= (m_{-1}/s)+m_0+m_1 s+\ldots \end{aligned} \tag{2.6}$$

一方、前節での方法によって得られたアドミタンスから計算された各ノードでの電圧値（s領域）とのsの係数を比較することによって k_j や p_j の値を求め電圧波形の式を得ることができる。これらの方法を使えば少ないパラメータで、電圧波形を表現でき、SPICE に比べて計算量を軽減することができる。
今回次の形の電圧波形を仮定した。

$$v(t)=Vdd\left(1-\exp\left(\frac{t-d}{\tau_2}\right)\right) \tag{2.7}$$

ここで、d及びτは moment matching により次のように計算される。

$$\begin{aligned} \tau_2 &= \sqrt{2m_1-m_0^2} \\ d &= m_0-\sqrt{2m_1-m_0^2} \end{aligned} \tag{2.8}$$

なお、Elmore delay は電圧波形を次の形に仮定したときの時定数と一致する。

$$v(t)=Vdd\left(1-\exp\left(\frac{t}{\tau_1}\right)\right) \tag{2.9}$$

$$\tau_1 = m_0 \quad \text{(Elmore delay)} \tag{2.10}$$

2.3. 計算方法の比較

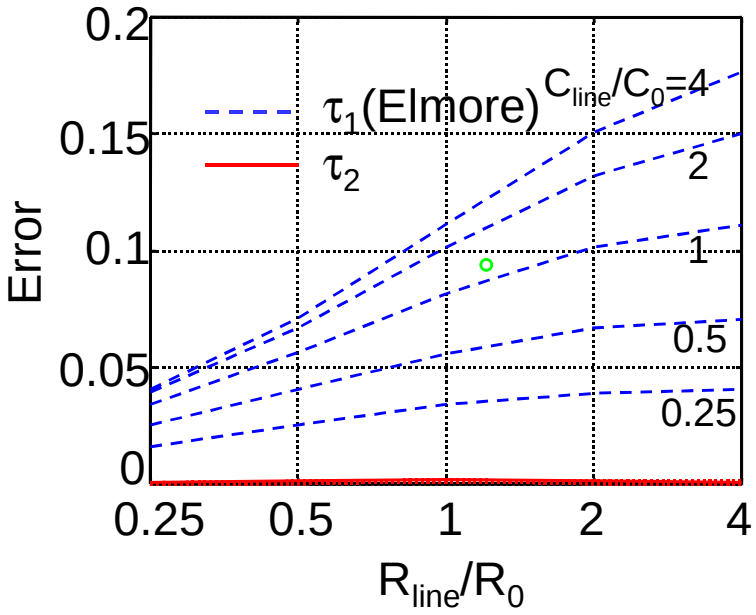

図 2.5　遅延モデルの違いによる誤差

2.1 の Elmore delay で示した方法と 2.2 で示した方法の比較、すなわち 2.7 式と 2.9 式の比較を行う。この節ではリピータを抵抗、容量で完全に近似した場合の、SPICE との誤差を考える。

リピータと配線の抵抗値や容量値の比を変化させたときの誤差を図に示した。図の中の丸で示した点はリピータ挿入に最適な抵抗値と容量値の比となる点である。グラフよりわかるように Elmore delay ではパラメータの変化に対して誤差が 20%近くまで変化する一方、今回 moment matching で計算したモデルでは誤差が１%以下となっている。特にリピータ挿入に最適な点の近傍では Elmore delay の誤差の変化が大きい。そのため今回、Elmore delay でなく式(2.7)のモデルを採用した。

2.4. リピータ等価抵抗値の決定

実際のトランジスタは線形性をもった抵抗ではないため、モデル式からの誤差が生じる。このため、トランジスタを等価抵抗で置き換える際の値が問題となる。今回計算に用いたリピータの等価抵抗を次のように決定した。

配線が十分に長いためリピータ数も多く、終端での効果が無視できる場合を考える。リピータを等間隔に同じサイズで挿入して(図 2.6)、SPICE を用いて遅延時間を計算する。このとき遅延時間は立ち上がり時間と立ち下り時間の平均値を用いる。

図2.7にリピータ間隔と信号伝達速度の関係を SPICE の結果とモデル式で計算した結果で示す。横軸はリピータ間隔であり、縦軸は長さを遅延時間で割った伝達速度(L/D)である。リピータサイズを1から 10 まで変化させてある。

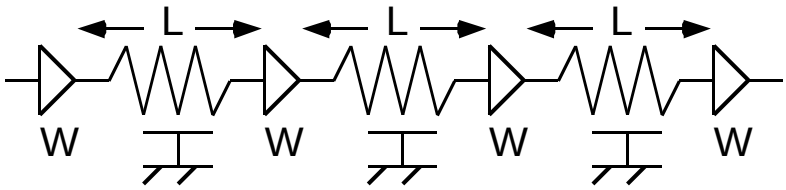

図 2.6　リピータ等価抵抗の決定法

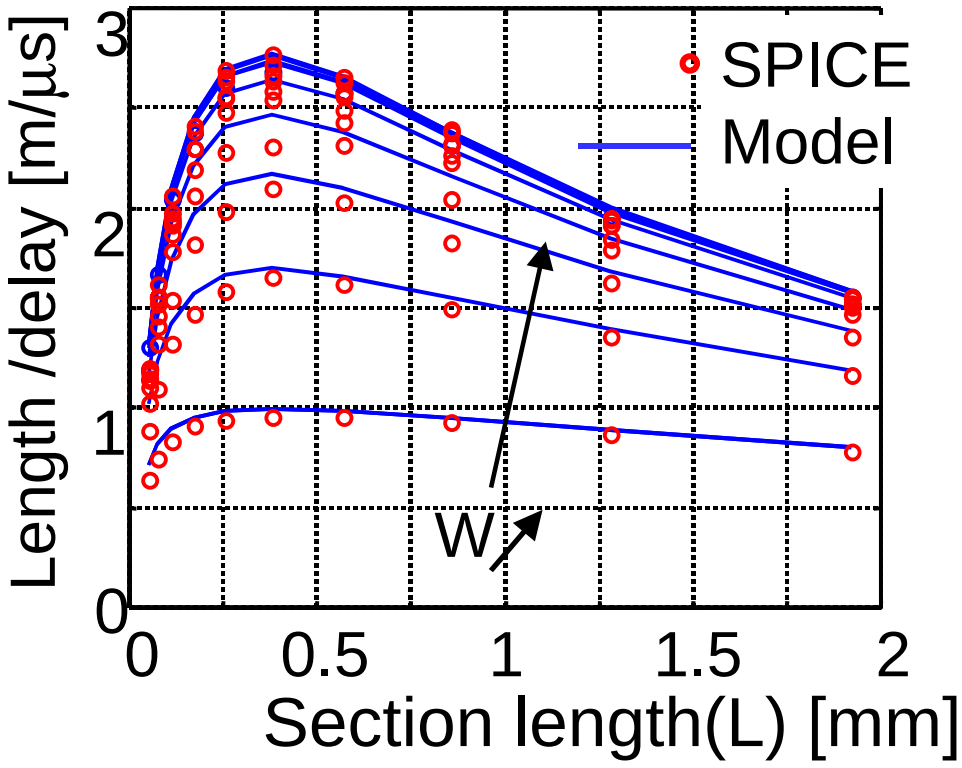

図 2.7　リピータ間隔と単位長さあたりの遅延時間

図 2.7 のグラフのなかで伝達速度の大きな値を用いれば遅延時間が最適となるよう設計できる。そのため、この点で遅延時間が一致するようにリピータの等価抵抗を決定すれば、モデル式の近似も良く、回路全体の遅延時間を最適化することが可能となる。

2.5. 遅延時間計算モデル

本研究での遅延時間モデルを示す。

出力端での電圧波形を次の指数関数の形に仮定する。

$$v(t) = Vdd\left(1 - \exp\left(\frac{t-d}{\tau_2}\right)\right) \tag{2.11}$$

このとき、時定数は、モーメントを用いて次のようにあらわされる。

$$\begin{aligned} \tau_2 &= \sqrt{2m_1 - m_0^2} \\ d &= m_0 - \sqrt{2m_1 - m_0^2} \end{aligned} \tag{2.12}$$

この時定数を用いて遅延時間を評価する。すなわち、入力端から出力端までの各セクションでの時定数の和が最小になるように、リピータ挿入を行う。この式の妥当性については次章で論ずることとする。

SPICE での遅延時間は入力端の電圧値が供給電圧の 50%に達した点から、出力端の電圧値が供給電圧の 50%に達するまでの時間をその出力端での遅延時間と定義する。また、SPICE での分布定数線路はπ型線路を 10 段接続することにより実現してある。

回路全体の遅延時間は全ての出力端での遅延時間を計り、その中の最大値で定義する。すなわち図 2.8 のような回路での遅延時間は

$$Delay = \max(D1, D2, D3) \tag{2.13}$$

となる。

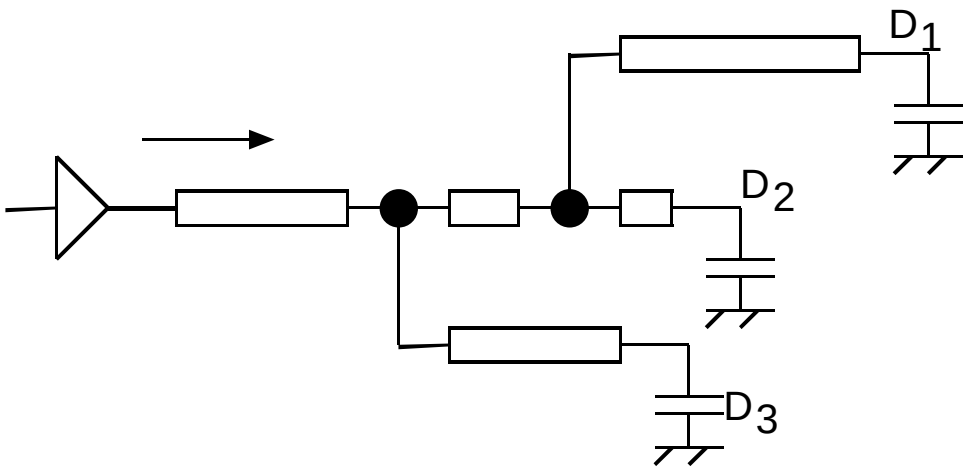

図 2.8　ツリー構造配線

第3章　　遅延時間最適化リピータ挿入法

3.1. 等間隔リピータ挿入法

最も単純なリピータ挿入方法を考える。この節で示す等間隔リピータ挿入法は一様な線路をリピータ挿入によって最適化し、それらを組み合わせることによって生成される。挿入するリピータは等間隔かつ、同サイズとする。入力端の駆動力が小さい場合には cascaded driver を挿入する。分岐部分については、分岐後の線路の先頭に最適サイズのリピータを挿入するかあるいは挿入しない。この方法はこの章で示す 3 つの方法の中で最も簡単な挿入方法である。

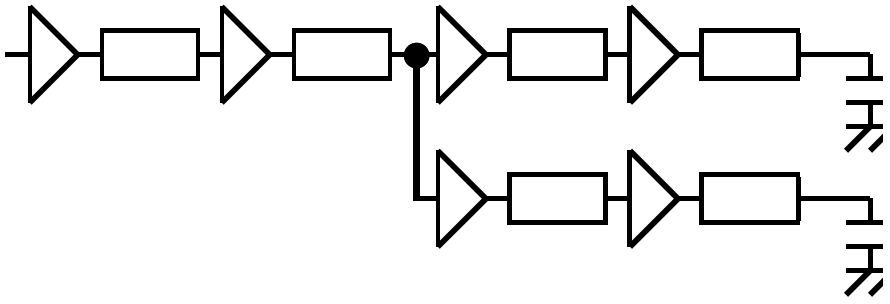

図 3.1 等間隔リピータ挿入法

この方法で挿入した場合、次の問題点があると考えられる。

1．出力端の容量によらず等間隔に挿入するため、最後のリピータの間隔が最適でない。
2．分岐部分での最適化が行われていない。

3.2. Simulated diffusion によるリピータ挿入最適化法

3.2.1. Simulated annealing 及び simulated diffusion について

一本の線路について遅延を最小化する問題については 1 章で示したように等間隔に最適サイズのリピータを挿入すればよい。しかし、これを実際の問題に適用するときは、複雑な枝分かれ線路での最適化問題となり、問題は複雑化する。 今回、このような repeater 挿入の問題をとくために simulated diffusion と呼ばれる方法を用いて最適化をおこなった。まず simulated annealing について説明し、その後それを改良した simulated diffusion を説明する。

Simulated annealing[6]は Kirkpatirick によって発表されたアルゴリズムであり最初は自動配線プログラムに適用された。このアルゴリズムは最適化の一般的な手法となっているためいくつかの分野に適用されていて、良い決定論的なアルゴリズムがない場合に対して有効な解を示すことが知られている。

アルゴリズムを図 3.2 に示す。物質を熱し徐々に冷却すると全体のエネルギーが最小化されるという焼きなまし（annealing）の現象をソフトウェアでまねるものである。新しい状態について評価関数を評価し、その結果が前の状態よりも良ければその状態を採用し現在の状態とする。しかし、前の状態よりも評価関数が悪化した場合もある確率でその状態を採用するところがこのアルゴリズムの特徴である。悪化の度合いが大きいほど採用する確率は指数関数的に減少する。計算の初期では悪くても採用する確率を大きくしておき計算時間とともに徐々にその確率を小さくしていく。これはちょうど系の温度 T を徐々に減少することに対応する。

この方法を用いることにより局所的な極小解（local minimum）に陥ることなく大局的な最適解を探索することができる。

一方、simulated diffusion[7] は連続的な解をもつ場合の最適化手法であり、MOSFET の model parameter へのフィッテング手法等に用いられている。Simulated diffusion では新しい状態を作り出す方法が 2 通りある。一方はその状態近傍での評価関数から 2 次近似したときの極小点を予想し新しい状態とするものでありこれが simulated diffusion の特徴となっている。これは分子がポテンシャル

の gradient 方向に移動することに対応する。またもう一方の状態生成の方法はある確率分布に従った擾乱によって新しい状態を生成する方法である。これらの状態生成方法により simulated annealing よりも高速に解を得ることができる。

Simulated annealing や simulated diffusion は本質的な部分は単純であるが、新しい状態の生成方法、評価関数の選定、冷却方法、初期状態等をそれぞれの問題に応じて最適化しなくてはならない。

```
温度 T =初期値
状態 X =初期値
while( 収束するまで ){
        for( 一定回数 ){
            新しい状態 Y を生成
            Δ E = E(Y) – E(X)
            if( Δ E < 0 ) then X = Y
            else{
                P = exp( - Δ E / T )
                R= ( 0, 1) の乱数
                if(R < P) then X = Y
            }
        }
        温度冷却
}
```

図 3.2 Simulated annealing の基本アルゴリズム

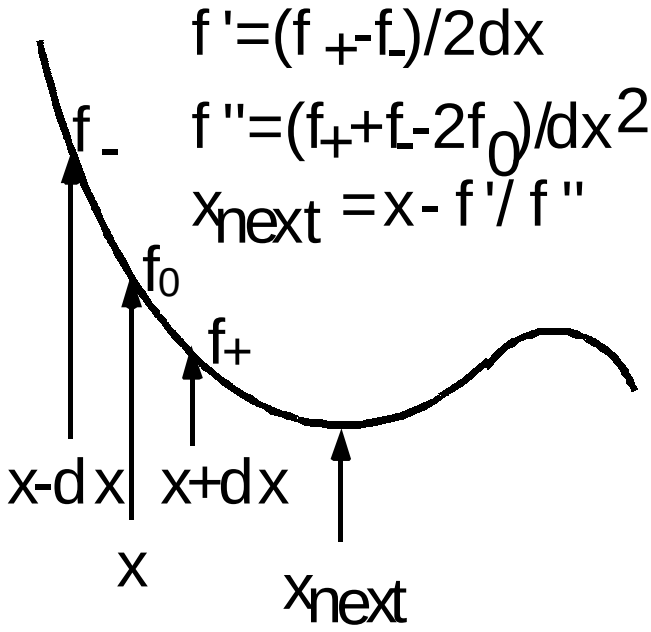

図 3.3　Simulated diffusion の微分情報を用いた状態生成法

3.2.2.　　変数の設定

今回行ったリピータ挿入法では、各配線区間毎に次の量を変数に設定した。

・リピータ数(k)

・リピータの位置(h[i])　（i = 1～k ）

・リピータのサイズ(x[i])　（i = 1～k ）

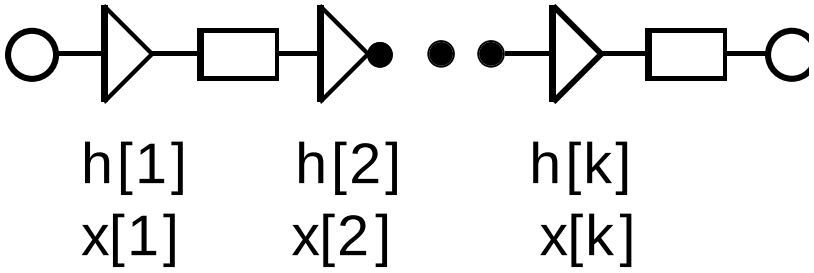

図 3.4　変数の設定方法

全てのリピータについて位置、及びサイズを変数に設定した。すなわち、リピータ数が k 個であれば、(2k)変数関数の問題であり、k もまた変数である。

3.2.3. 状態生成法

リピータの位置、大きさについては連続的な変数であるため微分情報を用いた状態生成法を用いるのが効果的である。状態生成の方法を次のようにした。

乱数ｊ, ｋ を発生させて、配線[j]の中のリピータ[k]について次の操作を行い新たな状態を生成した。

1. ランダムな状態生成
    1．1 リピータの数を1つ増加
    1．2 リピータの数を1つ減少
    1．3 リピータ[k]を1つとって、1つ増加
    1．4 リピータ[k]サイズを小さくする
    1．5 リピータ[k]サイズを大きくする
    1．6 リピータ[k]位置を入力端方向に動かす
    1．7 リピータ[k]位置を出力端方向に動かす
2. 微分情報を用いた状態生成
    2．1 リピータ[k]サイズを最適化
    2．2 リピータ[k]位置を最適化

3.2.4. 終了条件

探索の終了条件（図３．２の while 文の終了条件）を記す。

１．この while 文での一回目のループでの温度設定は０（すなわち改善された場合しか採用しない）、また一回目のループ終了後に得られたリピータ挿入での遅延時間の 10 分の１を次のループからの温度に設定した。また、冷却は毎回、温度を 0.9 倍することにより実現した。

２．探索中に得られた最良の解を保存しておき一定回数の探索の結果、状態の改善が十分小さく、最後の状態よりもいい状態を探索中に発見している場合は、最良であった状態に戻って探索を行う（while 文の先頭に戻る）。

３．状態の改善（リピータ挿入法の改善）があった場合はその差分を保存する。その差分の最大値が十分小さくなり、現在の状態が今まで探索した中での最良の状態であったとき（すなわち２の条件を満たさないとき）探索を終了する。

3.3. 新たなリピータ挿入法

単純なアルゴリズムでの問題点は、
１、出力端の容量によるリピータ最適位置の変化
２、分岐部分でのリピータ挿入法
が問題であった。そのためこれらを改善するリピータ挿入方法を考える。

１の問題については、Elmore delay　を計算し、その式より最も出力端よりのリピータの位置を決定し、その結果そのリピータが入力端側にずれて、リピータ数が冗長ならばリピータ数を減らすことにより解決した。

２の問題については、分岐後のリピータのサイズを変化させることにより対応した。一般的に、分岐後は容量、抵抗値が突然増加して見えるため、分岐の直後にリピータを挿入する。そのサイズを最適化することにより分岐部分でのリピータ挿入法を決定した。

3.3.1. 全体図

リピータ挿入の全体の流れは次のようになる。

１、配線、トランジスタのパラメータの計算。各回路負荷容量の計算
２、リピータサイズ、リピータ間隔の決定
３、各配線区間での最も時間のかかる出力端までの遅延時間の見積もり（概算）
４、線路毎にリピータ挿入パラメータ決定

１のパラメータ計算は、配線については配線の厚さ、幅、、長さ、等のパラメータから決定する。リピータ等価抵抗については 2.4 でその方法を示した。リピータ容量についてはゲート容量、接合容量から最終的には SPICE によるシミュレーションで決定する。

２の最適リピータサイズ、最適リピータ間隔の決定は 1.1 で示した式を用いた。

３の遅延時間の見積もりは、各配線区間で配線抵抗 R_{line}、 配線容量 C_{line} のとき遅延時間を

$$D = \sqrt{R_{line} C_{line}} \qquad (3.1)$$

とし、最も時間のかかる出力端までの時間を計算する。なお、この遅延時間は境界条件を無視した場合、同じリピータパラメータのもとではこの値に比例することになる。この概算した遅延時間は回路分岐部分での先頭リピータのサイズを決定するときに必要となる値である。

４のリピータ挿入パラメータ決定法は次節 3.3.2 で説明を行う。

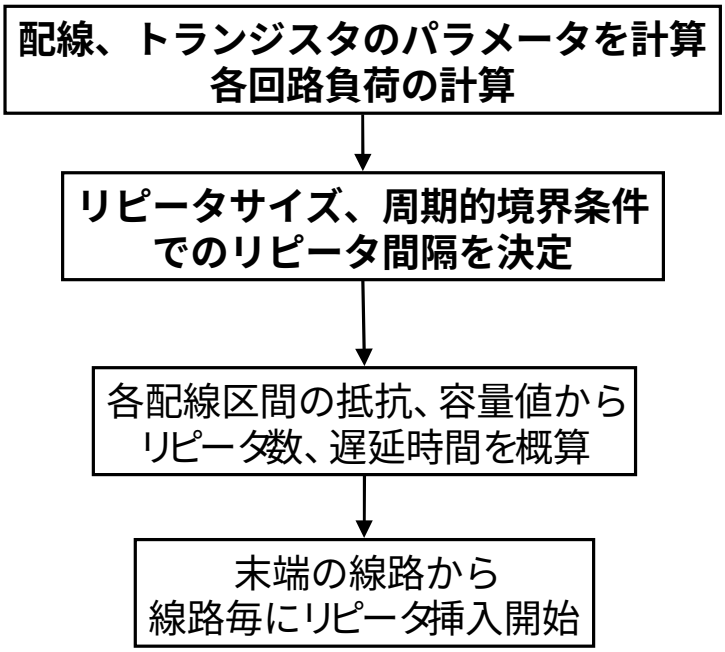

図 3.5　全体の流れ

3.3.2.　　リピータ挿入位置決定法

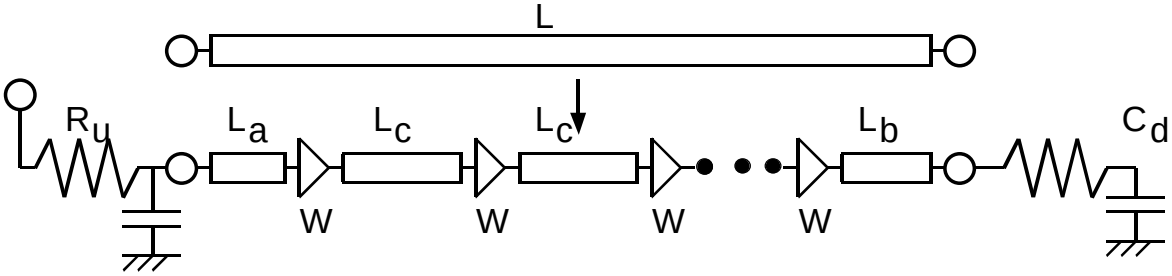

図 3.6　リピータ挿入法

図 3.6 の様な回路中の線路について考える。線路の入力端方向に抵抗成分を Ru として、線路の出力端方向の容量成分を Cd とする。なお、入力端方向の容量成分、出力端方向の抵抗成分は Elmore delay に影響を与えないため考えないこととする。また、途中に挿入するリピータを全て同じ大きさ(W)で、等間隔(Lc)に挿入するとする。

Elmore delay を考えた時、遅延時間を La 及び Lb で微分することにより、La 及び Lb の最適値は次のように求められる。

$$L_a = L\left(\frac{1}{n+2} - \frac{R_u}{R_{int}}\frac{n+1}{n+2} + \frac{r_t}{R_{int}W}\frac{n+1}{n+2} - W\frac{c_t}{C_{int}}\frac{1}{n+2} + \frac{C_d}{C_{int}}\frac{1}{n+2}\right) \qquad (3.2)$$

$$L_b = L\left(\frac{1}{n+2} + \frac{R_u}{R_{int}}\frac{1}{n+2} - \frac{r_t}{R_{int}W}\frac{1}{n+2} + W\frac{c_t}{C_{int}}\frac{n+1}{n+2} - \frac{C_d}{C_{int}}\frac{n+1}{n+2}\right) \qquad (3.3)$$

この式を用いて次のように各配線についてリピータ挿入の位置を決定する。また、リピータを挿入する順番は遅延時間（概算値）の大きな配線の出力端方向から行う。そのため、出力端方向の配線のリピータ挿入についてはどの段階でも完了していることになる。

1、出力端方向に向かって、容量を計算する。

2、リピータ最適位置を式より計算する。

3、リピータ間隔を調べ、それが理想的なリピータ間隔よりも小さければ、リピータ数が冗長であるから、リピータ数を１つ減らして、２にもどる。

4、リピータ位置を決定し、挿入する。

３の過程があることにより、出力端の容量が小さいことによって生じる冗長なリピータを削減することができる。

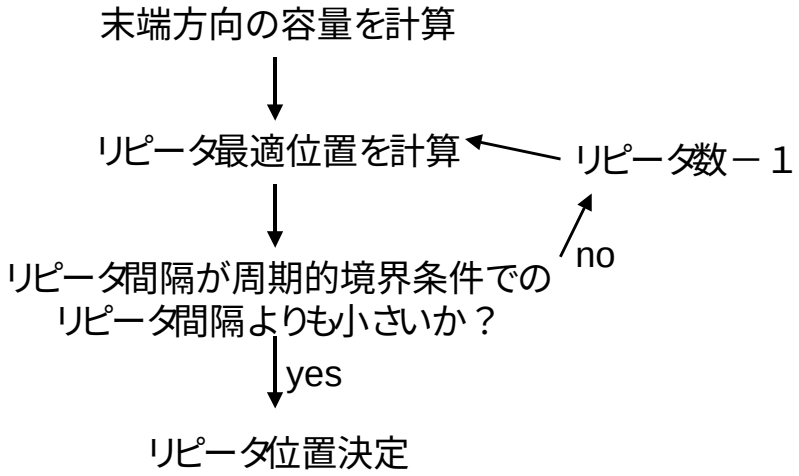

図 3.7　各線路毎のリピータ挿入法

3.3.3.　分岐部分のリピータ挿入法

分岐部分のリピータ挿入法については、次のように先頭のリピータサイズを決定することによって最適化を行う。

1．前段のリピータが小さいとき、すなわち駆動力が足りなければ、cascaded 接続を行う。
2．分岐した後の線路が共に短く、分岐の前にリピータを挿入すると結果的に遅くなる場合は

先頭部分へのリピータ挿入を行わない。

３．１及び２の条件を満たさなければ次の計算式によってWを計算する。

$$W = W_{opt} \frac{D_1}{D_1 + D_2} \qquad (3.4)$$

（ここで D_1 とは分岐後、その線路を通って出力端に達するまでの概算した遅延時間であり、D_2 とは、分岐後別の線路を通って出力端に達するまでの概算した遅延時間である。）

４．W が最小サイズより小さければ最小サイズリピータとする。

５．分岐後の線路の他方が最小サイズリピータならば、最適サイズリピータを挿入する。

６．４及び５の条件を満たさなければ先頭リピータサイズをWとする。

分岐後の線路が短く、その線路がその回路の遅延時間に影響しないときは分岐後に挿入するリピータのサイズを最小サイズにすることにより、他の出力端への遅延時間を小さくすることができる。しかし分岐後の線路の長さが同程度の場合、いずれかを最小サイズのリピータを挿入した場合、その挿入されたほうの遅延時間が増大するのは明白である。そのため、(3.4)式を用い、同程度の長さのときは同程度のリピータサイズになるようにした。また、リピータサイズが最適サイズ（先頭以外のリピータと同じサイズ）の場合、式(3.3)によって位置の最適化を行う。

分岐後の線路がいずれも短い場合、挿入することによって遅延時間が増す場合がある。そのために２の過程がある。分岐前にリピータがあった場合の遅延時間を計算し、その場合に遅延時間が大きくなるばあいあは、リピータをいずれの線路にも挿入しない。いずれかが十分（リピータ挿入が必要なほど）長い場合は、短いほうの線路には、分岐後の容量が大きい場合は挿入し、小さい場合は挿入しない。

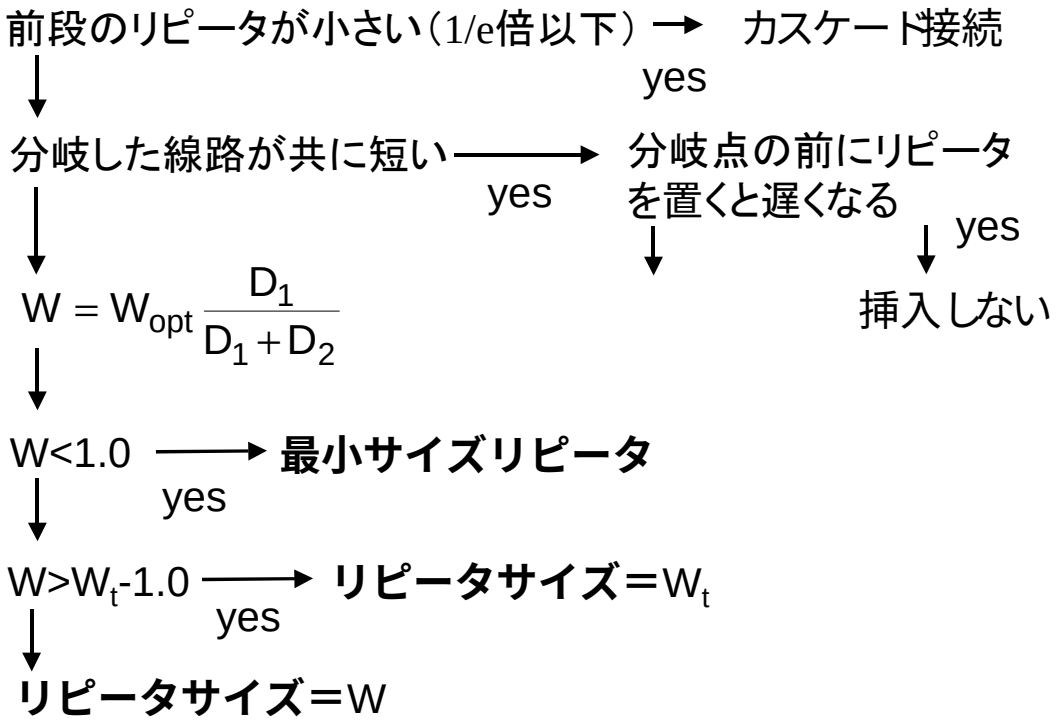

図 3.8　先頭リピータサイズ決定法

3.4. リピータ挿入結果比較

配線の長さをランダムに変化させ、それらを組みあわせた回路を発生し、それらに対するリピータ挿入の解を 3 つの方法で調べ、比較を行った。回路の長さの対数が一様に分布するように設定したた。そのためリピータ挿入の必要のない配線区間の出現する割合は大きい。

図 3.9、図 3.10 に fanout=3 の結果を示す。図 3.9 はリピータ挿入の結果、遅延時間をモデル式で計算したものであり、図 3.10 はリピータ挿入の結果、遅延時間を SPICE により検証したものである。遅延時間をモデル式で計算した場合は等間隔リピータ挿入法の遅延時間は simulated diffusion で最適化した場合に比べ、7.1%増加するのに対し今回提案した方法では 1.6%の増加となる。遅延時間を SPICE で計算した場合も simulated diffusion で最適化した場合に比べて、それぞれ 7.6%、1.6%の増加となった。すなわち、遅延時間計算方法の違いによるリピータ挿入の精度の変化は小さいことがわかる。

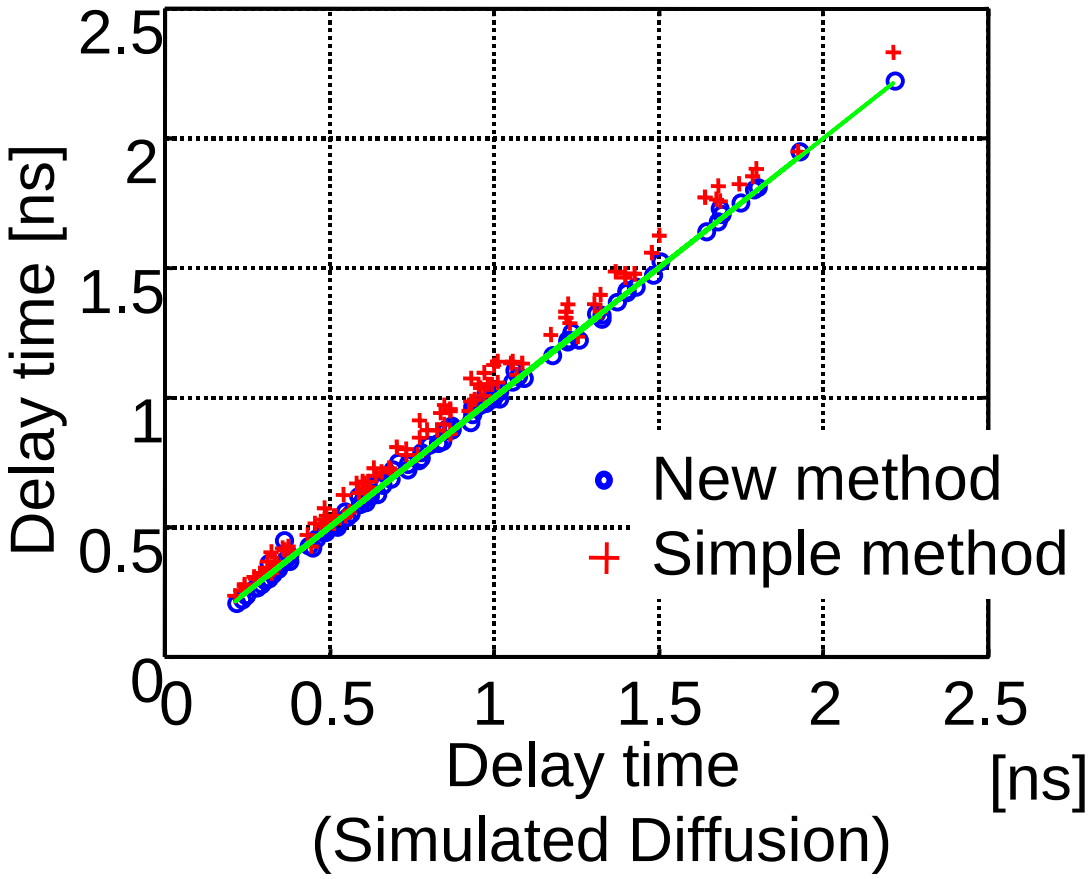

図 3.9 リピータ挿入結果比較 1

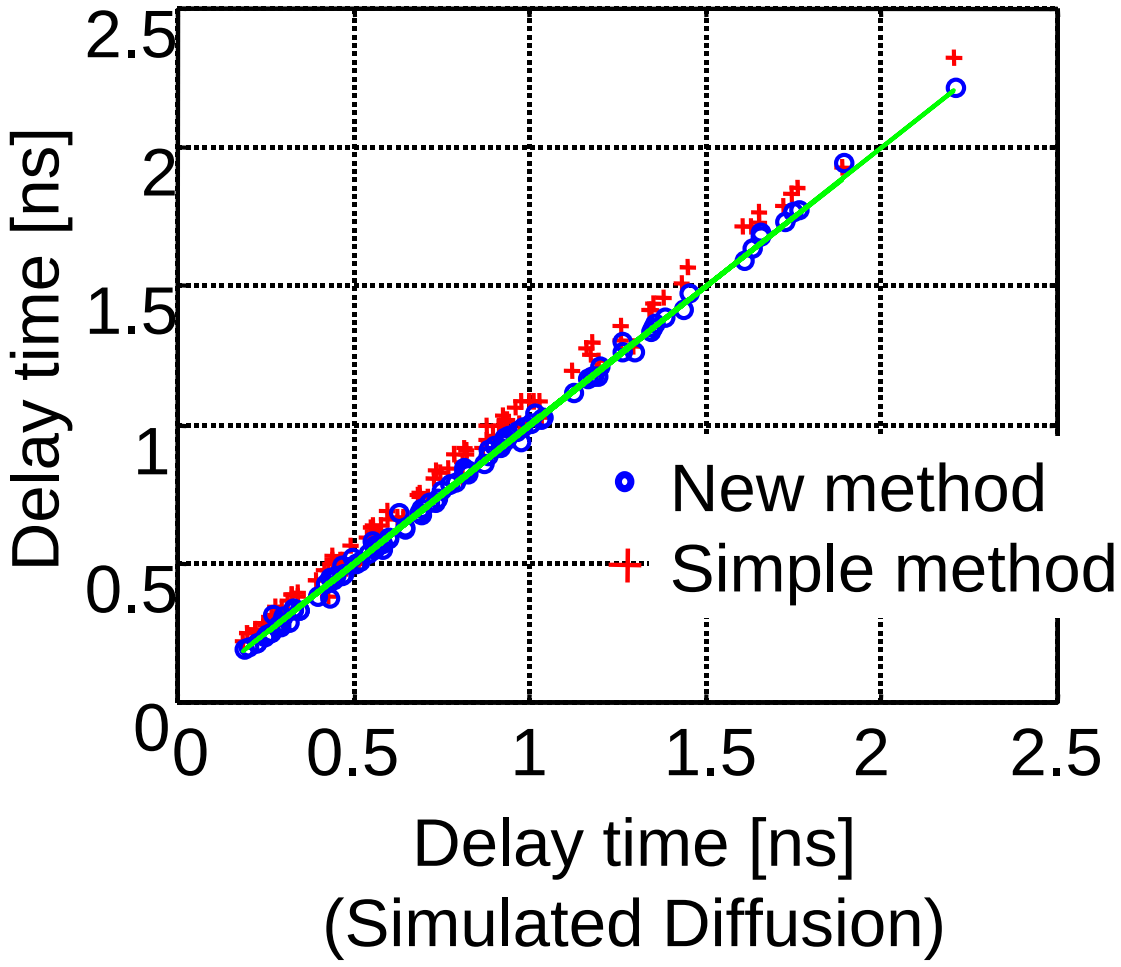

図 3.10　リピータ挿入結果比較 2

3.5. リピータ挿入法比較

上記 3.1、3.2、3.3 で示した３つのリピータ挿入法について、精度及び計算時間の点から比較検討を行う。

精度の点では simulated diffusion が最適であるが、今回新しく考えたリピータ挿入法でも平均 1.6%しか変わらない。等間隔リピータ挿入法では平均 7.6%遅延時間が増加した。

平均計算時間は、simulated diffusion では 1 つの回路あたりの計算時間が 7 秒であり、他の２つの方法に比べて計算時間が 200 倍以上となっている。実際の回路設計に利用することを考えると、配線遅延が問題となる部分が少ない場合には適用可能であるが、将来 LSI 全体の配線遅延が問題となり、チップ全体で 10 万回路程度の最適化が必要となった場合は 10 日と 1 時間の違いとなり、simulated diffusion は実際の設計で使うことは困難となると考えられる。

	精度	平均計算時間
Simulated diffusion	#	7s
Simple method	+7.6%	30ms
New method	+1.6%	30ms

表２　リピータ挿入法の比較

3.6. 遅延時間モデル式の妥当性

計算に用いた遅延モデルの妥当性を、リピータ挿入結果を SPICE で実行することにより検証を行った。

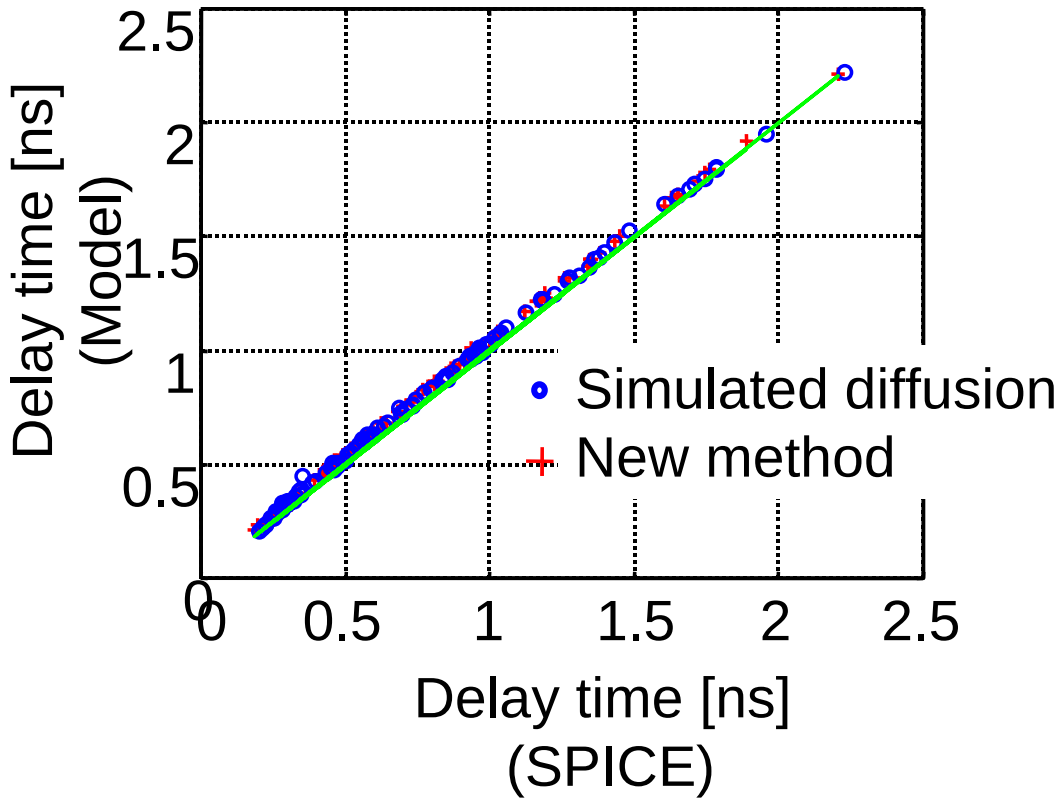

図 3.11　SPICE とモデル式の比較

リピータ挿入の結果を SPICE とモデル式の両方で計算した結果を図に示した。Simulated diffusion、new method、simple method３つの方法で挿入した結果全てにおいて、平均 5%程度の誤差となった。グラフよりわかるように遅延時間の小さい場合は誤差の割合が大きく、遅延時間の増大と共に誤差の割合が小さくなる。これはリピータ挿入の数が増加し、分岐部分での効果が小さくなるためと考えられる。また 3.4 節で示したようにこの誤差による、リピータ挿入法の精度の変化は小さく、いずれの方法でも平均 5%程度の誤差であった。

第4章　　低消費電力設計のためのリピータ挿入法

遅延時間を最適化するようにリピータを挿入する場合、リピータでの消費電力や面積は増大する。特に消費電力については、遅延時間最適化設計ではリピータ部分での容量が配線の 1.6 倍になるために、消費電力は著しく増大することになる。

そのため、この章では消費電力を考慮したリピータ挿入を考察する。

また、この章で扱う配線は全て一様な単一配線のみを考慮し、境界条件を考えないこととする。

4.1. 遅延時間と消費電力

配線が一様で十分長く（すなわち挿入リピータ数が多く）、終端条件を無視できる場合を考える。Elmore による遅延モデル式は次のようになる。

$$\tau_1 = R_0\left(C_0 + C_{line}\right) + R_{line}\left(C_0 + \frac{1}{2}C_{line}\right) \qquad (4.1)$$

リピータ数K、リピータサイズWのときのElmoreによる遅延時間のモデル式は、リピータ容量がサイズに比例し、リピータ等価抵抗がサイズに反比例するので

$$\begin{aligned}\tau_1 &= K\left\{\frac{R_0}{W}\left(WC_0+\frac{C_{line}}{K}\right)+\frac{R_{line}}{K}\left(WC_0+\frac{1}{2}\frac{C_{line}}{K}\right)\right\}\\ &= KR_0C_0+\frac{R_{line}C_{line}}{2K}+\frac{R_0C_{line}}{W}+WR_{line}C_0\end{aligned} \tag{4.2}$$

であり、K及びWの最小値は

$$K_0=\sqrt{\frac{R_{line}C_{line}}{2R_0C_0}},\quad W_0=\sqrt{\frac{R_0C_{line}}{R_{line}C_0}} \tag{4.3}$$

となる。(4.3)式を用いることにより、Elmore での遅延式では、リピータ挿入の結果の遅延時間はK_0W_0を用いて次のように表すことができる。

$$\tau_1=\left\{\frac{W}{W_0}+\frac{W_0}{W}+\left(\frac{K}{K_0}+\frac{K_0}{K}\right)\frac{1}{\sqrt{2}}\right\}\sqrt{R_0C_0R_{line}C_{line}} \tag{4.4}$$

今回用いた遅延モデルでは、一様な線路での遅延時間(τ_2)は次のように表される。

$$\begin{aligned}\tau_2^2 &= \left\{R_0\left(C_0+C_{line}\right)+R_{line}\left(C_0+\frac{1}{2}C_{line}\right)\right\}^2\\ &-\left\{\frac{1}{2}R_0C_0R_{line}C_{line}+\frac{1}{6}\left(R_{line}^2C_0C_{line}+R_{line}R_0C_{line}^2\right)+\frac{1}{24}R_{line}^2C_{line}^2\right\}\end{aligned} \tag{4.5}$$

これにサイズWのリピータをK個挿入すると、

$$\begin{aligned}\tau_2^2 &= \left(KR_0C_0+\frac{R_0C_{line}}{W}+WR_{line}C_0+\frac{R_{line}C_{line}}{2K}\right)^2\\ &-K^2\left\{\frac{1}{2}R_0C_0\frac{R_{line}}{K}\frac{C_{line}}{K}+\frac{1}{6}\left(\frac{R_{line}^2}{K^2}WC_0\frac{C_{line}}{K}+\frac{R_{line}}{K}\frac{R_0}{W}\frac{C_{line}^2}{K^2}\right)+\frac{1}{24}\frac{R_{line}^2}{K^2}\frac{C_{line}^2}{K^2}\right\}\end{aligned} \tag{4.6}$$

となる。この(4.6)式に(4.3)式で定義されたW_0、K_0を導入することにより、

$$\tau_2=\sqrt{\left\{\frac{W}{W_0}+\frac{W_0}{W}+\left(\frac{K}{K_0}+\frac{K_0}{K}\right)\frac{1}{\sqrt{2}}\right\}^2-\left\{1+\frac{\sqrt{2}}{3}\left(\frac{W}{W_0}\frac{K_0}{K}+\frac{K_0}{K}\frac{W_0}{W}\right)+\frac{1}{6}\frac{K_0^2}{K^2}\right\}}\sqrt{R_0C_0R_{line}C_{line}}$$

(4.7)

と変形できる。

一方、配線とリピータでの消費電力(P)の和は次のようになる。消費電力が容量に比例する成分、すなわち充放電電流成分による消費電力のみを考えると、

$$\begin{aligned} P &\propto WKC_0 + C_{line} \\ &= \left(\frac{K}{K_0} \frac{W}{W_0} \frac{1}{\sqrt{2}} + 1 \right) C_{line} \end{aligned} \tag{4.8}$$

となる。

PD 積、つまり消費電力・遅延積で最小化する問題を考える。式(4.7)(4.8)より、

$$PD \propto \left(\frac{K}{K_0} \frac{W}{W_0} \frac{1}{\sqrt{2}} + 1 \right) \sqrt{\left[\left\{ \frac{W}{W_0} + \frac{W_0}{W} + \left(\frac{K}{K_0} + \frac{K_0}{K} \right) \frac{1}{\sqrt{2}} \right\}^2 - \left\{ 1 + \frac{\sqrt{2}}{3} \left(\frac{W}{W_0} \frac{K_0}{K} + \frac{K_0}{K} \frac{W_0}{W} \right) + \frac{1}{6} \frac{K_0^2}{K^2} \right\} \right]} \tag{4.9}$$

であるから、PD 積最小化の問題は(4.9)式を最小化する問題であることがわかる。また、式(4.7)、(4.9)の形よりわかるように、リピータ数、リピータサイズに対する消費電力や遅延時間の変化は配線やリピータのパラメータによらず与えられる。図 4.1 にリピータ部分での消費電力と遅延時間、消費電力遅延積(PD 積)をグラフで示した。

遅延時間最適ではリピータ部分での消費電力は配線部分の 0.60 倍である。すなわち消費電力が 1.6 倍に増加する。一方、PD 積最適設計ではリピータでの消費電力は配線での消費電力の 0.26 倍、すなわち消費電力が 1.26 倍となり、遅延時間は遅延時間最適設計のときと比べて 1.09 倍となっている。

遅延時間の最適値付近では、消費電力の増加に対して遅延時間の向上の割合が小さい。このため、リピータ数やリピータサイズを減らした低消費電力設計が有効であることがわかる。

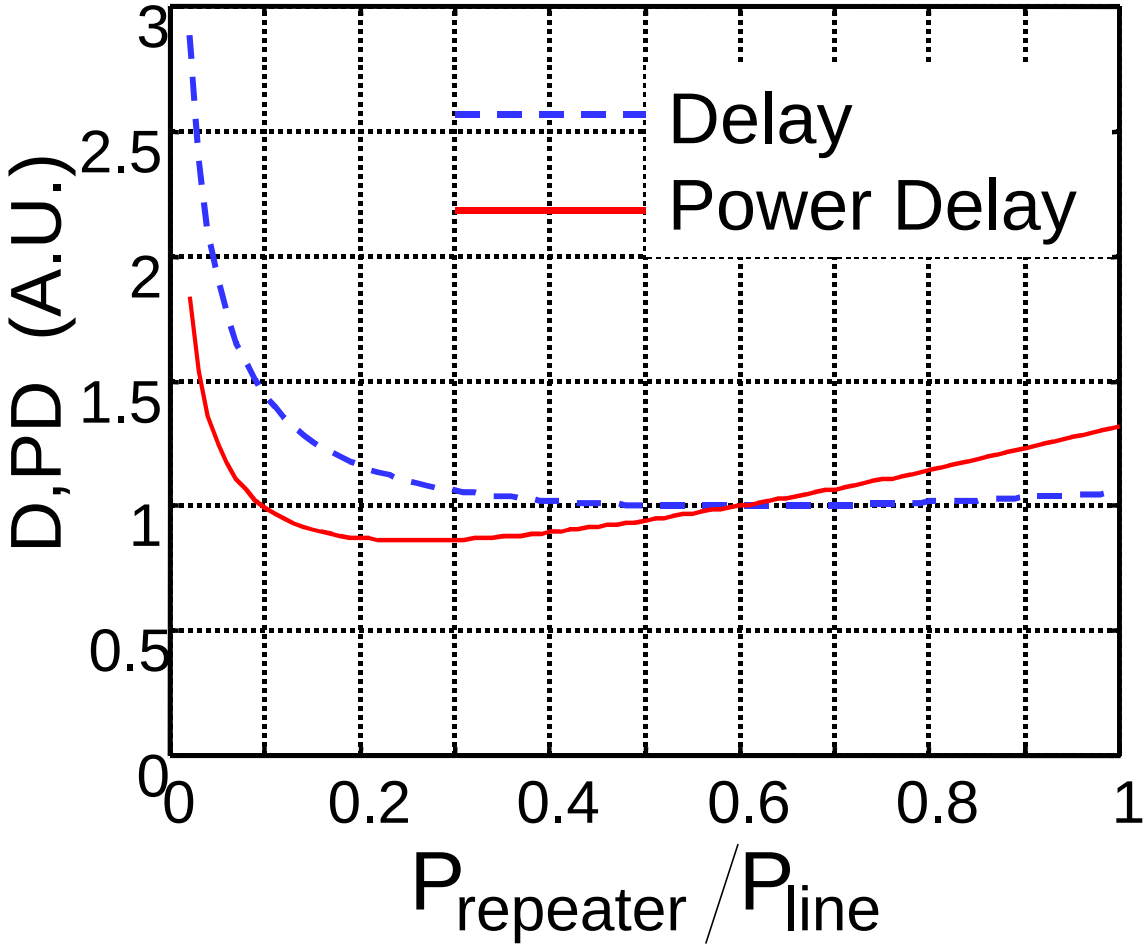

図４．１　リピータでの消費電力と遅延時間、消費電力遅延積の関係

4.2. 消費電力と最適リピータ数、最適リピータサイズの関係

(4.8)、(4.9)式より、リピータ部分での消費電力を設定したときのリピータサイズ(W)、リピータ数(K)の最適値が計算できる。図4．2に結果を示す。遅延時間最適のためには W/W_0 を 1.00 に、K/K_0 を 0.85 に設定すればよい。また、PD 積最適のためには W/W_0 を 0.72、K/K_0 を 0.51 に設定すればよい。

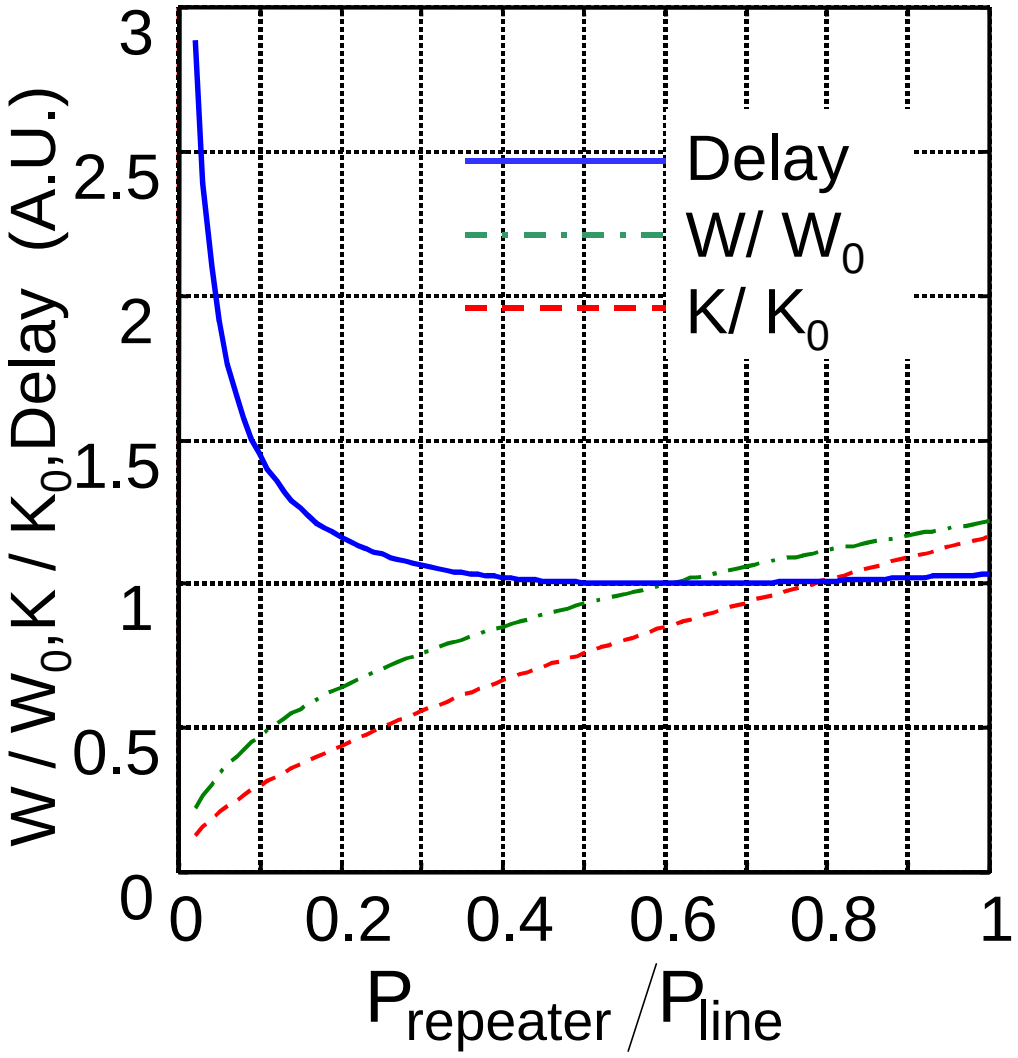

図4．2 リピータでの消費電力と最適リピータサイズ、最適リピータ数の関係

第5章　　結論

本研究で得られた結論を次に示す。

・リピータ挿入のための遅延時間計算法として Elmore delay では分布定数線路での誤差の変化が大きいため、改良した moment matching による遅延時間計算法を示した。また、リピータ等価抵抗の決定方法を示した。遅延時間計算法をもとにリピータ挿入を行い結果を SPICE で検証した。

・ツリー構造回路での遅延時間最適化のための等間隔にリピータを挿入する方法、simulated diffusion による方法、新たな提案としてのリピータ挿入の方法でのリピータ挿入を提案、計算し、新たなリピータ挿入法が高速かつ高精度なリピータ挿入法であることを実証した。

・消費電力、面積を考慮した上でのリピータ挿入の方法を単一線路の場合について考察し、テクノロジーに依存しない設計指針を示した。

・単一配線について、PD 積最小設計ののための方法を考え、遅延時間最小の場合に比べてリピータ数を 0.60 倍、リピータサイズを 0.72 倍にした結果、遅延時間は 1.1 倍、消費電力は配線部分での消費電力の 0.26 倍になることを示した。

第6章　　今後の課題

今回、ツリー構造回路についての遅延時間最適化を研究したが、今後リピータ挿入の研究には次のような課題が考えられる。

（1）位相

リピータは実際にはインバータで実現するため、リピータの数が奇数であれば位相は反転する。単一配線であれば最適リピータ数が奇数であったとき、リピータ数を一つ増加、あるいは減少することにより問題に対応できるが、これがツリー構造の場合は問題は複雑である。上位設計のレベルで対応することも考えられるが、リピータ挿入法の中で対応していくことが必要となるであろう。

（2）低消費電力設計のためのツリー構造回路でのリピータ挿入法

今回考えた配線は単一な配線であった。これをツリー構造でのリピータ挿入法を考える。Simulated diffusion の方法での実現は容易であるが、計算時間の少ない方法でのリピータ挿入の方法を考えなくてはならない。

Appendix プログラム

このプログラム（Simulated Diffusion 及び New algorithm ）は次のファイルよりなる。

line.h	ヘッダファイル
voltage.cc	電圧の計算
admittance.cc	アドミタンスの計算
delay.cc	遅延時間の計算
power.cc	消費電力の計算
awe.cc	Moment matching に関するもの
buffer.cc	リピータに関するもの
bufferio.cc	リピータの入出力に関するもの
netio.cc	ツリー構造回路のファイル入出力に関するもの
connection.cc	配線毎の接続に関するもの
SDfunc.cc	Simulated diffusion　に関する関数
printfunc.cc	入出力に関するもの
SA.cc	Simulated Diffusion
new.cc	New algorithm

```
/* line.h */

/*************************************************/
/*************************************************/
/*                  header file                  */
/*************************************************/
/*************************************************/

/*************************************************/
/* changed to use sink capacitance               */
/*             and driving buffer parameter      */
/*                                    99/1/11    */
/*************************************************/

#define DEBUG (1)

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <errno.h>


#if !defined(RAND)
#if defined(RAND_MAX)
#define RAND (1.0*rand()/RAND_MAX)
#else
#define RAND (1.0*random(65535)/65535)
#endif
#endif

/*************************************/
/* for debugging macro and variables */
/*************************************/

#ifdef DEBUG
#define DebugPrint(message) fprintf(stderr,message)
#define DebugPrint2(f,a,b)  fprintf(stderr,f,a,b)
#else
#define DebugPrint(message)
#define DebugPrint2(f,a,b)
#endif

/*********************************/
/*       compile option          */
/*********************************/
#define DELAY_MODEL (1) /* set 0 when using 50% delay model */
                        /* set 1 when using other tau delay model */

/*********************************/
/* global variable and its limit */
/*********************************/

const double INITIAL_CASCADE_SIZE  = M_E;
const int ELEMENT = 100;
const int MAX_BUFFER = 100;
const double ERROR_RATE = 1e-9; /* in math.cc file */
const int LINE_MAX = 12;    /* line number max for one circuit */



/***********/
/* classes */
/***********/

class Buffering{
 public:
        double  position;
        double size;
           double Bdelay;
           Buffering(void){
                Bdelay = 0;
        }
};

class Admittance{
 public:
        int flag;
           double y1,y2;
```

```
//          double y3;
            Admittance(double = 0,double = 0, double = 0);
};

class Voltage{
 public:
            int flag ;
          double v_1,v0,v1;
//          double v2;
            Voltage(double = 0 , double = 0, double = 0, double = 0);
};

class Vt{
 public:
            int moment;

            /* moment = 0 */
            double tau0;
            /* moment = 1 */
            double delay,tau;
            /* moment = 2 */
            double k1,k2,tau_1,tau_2;

            Vt(void);
};

class Line{
 private:
          double r; /* resistance */
          double c; /* capacitance */

            double sink_c_size ; /* sink capacitance size */
                                 /* if this line is not end, sink_c is set to zero */
            double input_r ; /* input transistor size */
                              /* if this line is not root, sink_c is set to zero */
            /* delay */

            double Ldelay;  /* if line end */
            double Bdelay;  /* if included buffer */
            double BBdelay; /* buffer to buffer delay */

            /* simple parameter */
 public: /* tmporary */
            double x1 , x2 , x3; /* position */
            double h0 , h1 , h2 , h3; /* size     */

            int k0 ;             /* the number of cascaded driver */
            int k1 ;             /* dividing number    */

            /* for AWE */
 public:
            Admittance head_admittance;
            Admittance first_buffer_admittance;
            Admittance last_buffer_admittance;
            Admittance tail_admittance;

          Voltage head_voltage;
            Voltage first_buffer_voltage;
            Voltage last_buffer_voltage;
            Voltage tail_voltage;

            Vt vt;
            Vt first_buffer_vt;

            /* buffers */


          Buffering buffer[MAX_BUFFER];

          int total_buffer;
            double rep_N; /* this is made for Optimization of buffer insertion */
            double optW;
            double optL;

            double optimized_delay;
            int top_buffer_insert;
private:
            /* functions */
```

```
        void Sort_buffer(void);
        double Calc_ave_size(void);
        double Calc_delay(void);
 public:
        Line *parent ;
        Line *right ,*left ;
        Line(double ,double ,Line* = NULL , Line* = NULL, Line* = NULL);
        Line(void);

        double Get_r(void) {return r;}
        double Get_c(void) {return c;}
        double Get_sink_c_size(void) {return sink_c_size;}

        double Get_Ldelay(void)  {return Ldelay;}
        double Get_Bdelay(void)  {return Bdelay;}
        double Get_BBdelay(void) {return BBdelay;}

        int Get_total_buffer(void) {return total_buffer;}
        double Get_first_buffer_position(void) {return buffer[1].position;}
        double Get_first_buffer_size(void) {return buffer[1].size;}
        double Get_last_buffer_position(void) {return buffer[total_buffer].position;}
        double Get_last_buffer_size(void) {return buffer[total_buffer].size;}

      void Add_buffer(void);
      void Del_buffer(void);
      void Equally_divide(int ,int );
      void Optimize_position(void);
      void Optimize_size(void);

        void Print_buffer(void);
        void Print_line(void);

      void Print_admittance(void);

        void Print_voltage(void);

        void Calculate_buffer_buffer_delay(void);

        void Set_rc( double, double, double );

        void Set_Ldelay(Line&,double);
        void Set_Bdelay(Line&,double);
        void Set_BBdelay(Line&,double);

        /* for power calculation */

        double Calculate_line_power(void);

        /* for SA */

        int Make_buffer_large(int);
        int Make_buffer_small(int);
        int  Move_buffer_right(int);
        int  Move_buffer_left(int);

        /* for SD */
        int Move_to_min_position(int);
        int Optimum_sizing(int,double);
        int Optimum_positioning(int,double );


        void Clear_head_flag(void);
        void Clear_tail_flag(void);

        /* bufferio.cc file */
        void Insert_buffer(double ,double );

        /* for simplified parameter */
        void Simplified_parameter_initialize(void);
        void Simplified_parameter_transfer(void);
        void Simplified_parameter_out(void);
        void Clear_buffer(void);
};


class Circuit{
 private:
        double Cdelay;
```

```
          double power;
 public:
        Circuit(void){
                Cdelay = power = 0;
                     total_line = 1;
        }
          int total_line;
          Line line[LINE_MAX];
          double Return_p(void)  {return power; }
        double Return_d(void)  {return Cdelay; }
        double Return_pd(void) {return Cdelay * power;}
        double Return_ed(void) {return Cdelay * Cdelay * power;}

          void Put_admittance(Admittance& , Admittance);

          void Calculate_circuit_admittance(Line&);
        void Calculate_admittance(Line&);

          void Calculate_voltage(Line&);

        void Get_line_end_admittance(Line& );
          void Get_line_start_voltage(Line&);

          void Connect_line(int ,int);


          void Calculate_circuit_delay(void);
          double Calculate_specified_node_delay(Line&);
          void Calculate_branch_node_delay(Line&);
          double Calculate_delay(Line*);

          void Calculate_circuit_power(void);

          void Makecircuit( int&, char* net_file_name = NULL);
          void Insert_buffer(char *);

          /* in bufferio.cc file */
          void Insertion_from_file(void);
          void Output_buffer(char* buffering_file_name = NULL);

          /* for simplified parameter */
          void Simplified_parameter_initialize(void);
};

/************************/
/*      functions       */
/************************/

void laplace_transform(Voltage,Vt&,int );
void Print_vt(Vt);

/* in math.cc files */
double get_delay_from_vt(Vt);
double tau_delay_model_delay(Vt);
double tau_model_delay(Vt);

double double_exp_model(Vt ,double);
double search_50_delay(Vt vt);

void Calculate_line_admittance(Admittance&, Admittance , double ,double );
void Calculate_root_voltage(Voltage&, Admittance, double = 1);
void Calculate_line_voltage(Voltage, Voltage&,Admittance, double ,double);

void Print_admittance(Admittance adm);

void Print_circuit(Circuit*);
void Waveout(Vt,double);


inline void Line::Set_Ldelay(Line& line , double delay){
          line.Ldelay = delay;
}

inline void Line::Set_Bdelay(Line& line , double delay){
          line.Bdelay = delay;
}

inline void Line::Set_BBdelay(Line& line , double delay){
          line.BBdelay = delay;
```

```
}


/* get delay when vt model is tau model */

inline double tau_model_delay(Vt vt){
        return(vt.tau0 * log(2.0));
}

/* get delay when vt model is tau_delay model */

inline double tau_delay_model_delay(Vt vt){
        return(vt.delay + vt.tau * log(2.0));
}

/* get voltage from vt parameter and time */

inline double double_exp_model(Vt vt,double t){
        return( 1 - vt.k1 * exp( - t /vt.tau_1) - vt.k2 * exp( -t / vt.tau_2));
}



/* get delay time from vt parameter */

inline double get_delay_from_vt(Vt vt){
        double delay;
#if(DELAY_MODEL == 0)
           if(vt.moment == 0){
                delay = tau_model_delay(vt);
        }else if(vt.moment == 1){
                delay = tau_delay_model_delay(vt);
           }
#endif
#if(DELAY_MODEL == 1)
           delay = vt.tau;
           /* delay = vt.tau * 1.3499;*/
           /* log( 3.3 - 0.6) - log( 3.3 - 2.6) */
#endif

#if(WAVEOUT)
                Waveout(vt,delay);
#endif

        return(delay);
}
```

```
/* voltage.cc */

/*********************************************************/
/*********************************************************/
/*            calculate voltage file                     */
/*********************************************************/
/*********************************************************/

#include "line.h"

extern double ctr;
extern double rtr;
extern double drivingbuffer;
extern Voltage input_voltage;

Voltage::Voltage(double v_1_,double v0_,double v1_, double v2_){
            v_1 = v_1_;
            v0    = v0_;
            v1    = v1_;
//            v2    = v2_;
              flag = 0;
 }

void Line::Print_voltage(void){
          printf("head voltage¥n");
          printf("v_1 = %e¥n",head_voltage.v_1);
          printf("v0 = %e¥n",head_voltage.v0);
          printf("v1 = %e¥n",head_voltage.v1);
//          printf("v2 = %e¥n",head_voltage.v2);
            if(total_buffer != 0){
                        printf("first buffer voltage¥n");
                        printf("v_1 = %e¥n",first_buffer_voltage.v_1);
                        printf("v0 = %e¥n", first_buffer_voltage.v0);
                        printf("v1 = %e¥n", first_buffer_voltage.v1);
//                      printf("v2 = %e¥n", first_buffer_voltage.v2);
            }
          printf("tail voltage¥n");
          printf("v_1 = %e¥n",tail_voltage.v_1);
          printf("v0 = %e¥n",tail_voltage.v0);
          printf("v1 = %e¥n",tail_voltage.v1);
//          printf("v2 = %e¥n",tail_voltage.v2);
            if(total_buffer != 0){
                        printf("last buffer voltage¥n");
                        printf("v_1 = %e¥n",last_buffer_voltage.v_1);
                        printf("v0 = %e¥n", last_buffer_voltage.v0);
                        printf("v1 = %e¥n", last_buffer_voltage.v1);
//                      printf("v2 = %e¥n", last_buffer_voltage.v2);
            }
}

void Circuit::Get_line_start_voltage(Line& line){
            if(line.parent == NULL){
                        Calculate_root_voltage(line.head_voltage,line.head_admittance, ::drivingbuffer);
            }else{
                        if( (line.parent)->tail_voltage.flag == 0){
                                    Calculate_voltage(*(line.parent));
                        }
                        line.head_voltage = (line.parent)->tail_voltage ;
            }
            line.head_voltage.flag = 1;
}

/* calculate voltage for lines without buffer */

void Circuit::Calculate_voltage(Line& line){

            Voltage h_voltage;
            double last_buffer_position ;
            double last_buffer_size;
            double first_buffer_position;

            double r,c;

            if( line.Get_total_buffer() == 0){

                        Get_line_start_voltage(line);

                        h_voltage = line.head_voltage;
```

```
                    r = line.Get_r();
                    c = line.Get_c();

                    Calculate_line_voltage(h_voltage, line.tail_voltage,line.tail_admittance, r,c);

            }else{
                    /* calculate head voltage */

                    Get_line_start_voltage(line);

                    /* calculate last buffer voltage */

                    last_buffer_size = line.Get_last_buffer_size();

                    Calculate_root_voltage(line.last_buffer_voltage, line.last_buffer_admittance,last_buffer_size);

                    /* calculate tail voltage */

                    last_buffer_position = line.Get_last_buffer_position();

                    r = line.Get_r() * (ELEMENT - last_buffer_position) / ELEMENT;
                    c = line.Get_c() * (ELEMENT - last_buffer_position) /ELEMENT;

                    Calculate_line_voltage(line.last_buffer_voltage, line.tail_voltage,line.tail_admittance, r,c);

                    /* calculate head voltage */

                    first_buffer_position = line.Get_first_buffer_position();

                    r = line.Get_r() * first_buffer_position / ELEMENT;
                    c = line.Get_c() * first_buffer_position /ELEMENT;

                    Calculate_line_voltage(line.head_voltage,   line.first_buffer_voltage,   line.first_buffer_admittance,
r,c);
            }
}


void Calculate_line_voltage(Voltage h_voltage, Voltage &t_voltage, Admittance t_admittance, double line_R, double
line_C){

            double r,c;
            double t1,t2,t3;
            double a1,a2,a3;
            double b_1,b0,b1,b2;

            r = line_R;
            c = line_C;

            b_1 = h_voltage.v_1;
            b0   = h_voltage.v0;
            b1   = h_voltage.v1;
//          b2   = h_voltage.v2;

            a1 = t_admittance.y1;
            a2 = t_admittance.y2;
//          a3 = t_admittance.y3;


          t1 = -(a1*r) - (c*r)/2.;
          t2 = -(pow(c,2)*pow(r,2))/24.0 + pow(a1*r + (c*r)/2.,2) -
                  r*(a2 + (a1*c*r)/6.);
/*          t3 = -(pow(c,3)*pow(r,3))/720. -
                  r*(a3 + (a2*c*r)/6. + (a1*pow(c,2)*pow(r,2))/120.0) -
                  (a1*r + (c*r)/2.)*(-(pow(c,2)*pow(r,2))/24.0 +
                  pow(a1*r + (c*r)/2.,2) - r*(a2 + (a1*c*r)/6.)) +
                  (a1*r + (c*r)/2.)*((pow(c,2)*pow(r,2))/24.0 +
                  r*(a2 + (a1*c*r)/6.));
*/

          t_voltage.v_1 = b_1;
          t_voltage.v0   = t1 * b_1 + b0;
          t_voltage.v1   = t2 * b_1 + t1 * b0 + b1;
//          t_voltage.v2   = t3 * b_1 + t2 * b0 + t1 * b1 + b2;

            t_voltage.flag = 1;
}
```

```
void Calculate_root_voltage(Voltage& voltage, Admittance admittance, double buffer_size){
        double a1,a2,a3;
        double b_1,b0,b1,b2;
        double t1,t2,t3;

        double rtr;

            rtr = ::rtr/ buffer_size;

        a1 = admittance.y1;
        a2 = admittance.y2;
//          a3 = admittance.y3;

        t1 = -rtr * a1;
        t2 = rtr * rtr * a1 * a1 - rtr * a2 ;
//          t3 = 2.0 * rtr * rtr * a1 * a2 - rtr * a3;

        b_1 = ::input_voltage.v_1;
        b0  = ::input_voltage.v0;
        b1  = ::input_voltage.v1;
//          b2  = ::input_voltage.v2;

        voltage.v_1 = b_1;
        voltage.v0  = t1 * b_1 + b0;
        voltage.v1  = t2 * b_1 + t1 * b0 + b1;
//          voltage.v2  = t3 * b_1 + t2 * b0 + t1 * b1 + b2;

            voltage.flag = 1;

}
```

```
/* admittance.cc */
/*******************************************************/
/*******************************************************/
/*              calculate admittance file              */
/*******************************************************/
/*******************************************************/

#include "line.h"

extern double ctr;
extern Voltage input_voltage;
/* admittance constructer */

Admittance::Admittance(double y1_,double y2_    ,double y3_){
                y1 = y1_;
                y2 = y2_;
//              y3 = y3_;
                flag = 0;
}

/* calculate of add for admittance    */

void Add_adm(Admittance& total,Admittance ya, Admittance yb){
                total.y1 = ya.y1 + yb.y1;
                total.y2 = ya.y2 + yb.y2;
//              total.y3 = ya.y3 + yb.y3;
}

/*******************************************/
/* print admittance for debugging          */
/*******************************************/

void Print_admittance(Admittance adm){
        printf("y1 = %e¥n",adm.y1);
        printf("y2 = %e¥n",adm.y2);
//        printf("y3 = %e¥n",adm.y3);
}

void Line::Print_admittance(void){
        printf("head admittance¥n");
        printf("y1 = %e¥n",head_admittance.y1);
        printf("y2 = %e¥n",head_admittance.y2);
//          printf("y3 = %e¥n",head_admittance.y3);
            if(total_buffer != 0){
                        printf("first buffer admittance¥n");
                        printf("y1 = %e¥n",first_buffer_admittance.y1);
                        printf("y2 = %e¥n",first_buffer_admittance.y2);
//                      printf("y3 = %e¥n",first_buffer_admittance.y3);
            }
        printf("tail admittance¥n");
        printf("y1 = %e¥n",tail_admittance.y1);
        printf("y2 = %e¥n",tail_admittance.y2);
//          printf("y3 = %e¥n",tail_admittance.y3);
            if(total_buffer != 0){
                        printf("last buffer admittance¥n");
                        printf("y1 = %e¥n",last_buffer_admittance.y1);
                        printf("y2 = %e¥n",last_buffer_admittance.y2);
//                      printf("y3 = %e¥n",last_buffer_admittance.y3);
            }
}

void Circuit::Get_line_end_admittance(Line& line){

            Admittance right_adm,left_adm;

            if(( line.right == NULL) && ( line.left == NULL)){
                        line.tail_admittance.y1 = line.Get_sink_c_size() * ::ctr;

            }else if( line.right == NULL){
                        if((line.left)->head_admittance.flag == 0){
                                    Calculate_admittance(*(line.left));
                        }
                        left_adm = (line.left)->head_admittance;
                        line.tail_admittance = left_adm;

            }else if( line.left == NULL){
                        if( (line.right)->head_admittance.flag ==0){
                                    Calculate_admittance(*(line.right));
                        }
```

```
                right_adm = (line.right)->head_admittance ;
                line.tail_admittance = right_adm;

        }else{

                if((line.right)->head_admittance.flag == 0){
                        Calculate_admittance(*(line.right));
                }
                if((line.left)->tail_admittance.flag == 0){
                        Calculate_admittance(*(line.left));
                }
                right_adm = (line.right)->head_admittance ;
                left_adm = (line.left)->head_admittance;

                Add_adm(line.tail_admittance, ¥
                        right_adm , left_adm);
        }

        line.tail_admittance.flag = 1;
}

/*****************************************************/
/*          calculate admittance for a line          */
/*****************************************************/

void Circuit::Calculate_admittance(Line& line){

        Admittance t_admittance;
        double first_buffer_position ,last_buffer_position;
        double first_buffer_size        ,last_buffer_size;
        double r,c;

        if(line.Get_total_buffer() == 0){

                Get_line_end_admittance(line);

                t_admittance = line.tail_admittance;

                r = line.Get_r();
                c = line.Get_c();

                Calculate_line_admittance(line.head_admittance,line.tail_admittance, r,c);

        }else{
                /* calculate tail admittance */

                Get_line_end_admittance(line);

                /* calculate first buffer admittance */

                first_buffer_position= line.Get_first_buffer_position();
                first_buffer_size= line.Get_first_buffer_size(); ;

                t_admittance.y1 = ::ctr * first_buffer_size;
                t_admittance.y2 = 0;
//              t_admittance.y3 = 0;

        line.first_buffer_admittance = t_admittance;

        /* calculate head admittance */

                r = (line.Get_r() * first_buffer_position) / ELEMENT;
                c = (line.Get_c() * first_buffer_position)/ ELEMENT;

                Calculate_line_admittance(line.head_admittance,         t_admittance, r,c);

                /* calculate last buffer admittance */

                last_buffer_position = line.Get_last_buffer_position();
                last_buffer_size = line.Get_last_buffer_size();

                t_admittance = line.tail_admittance;

                r = (line.Get_r() * (ELEMENT - last_buffer_position)) / ELEMENT;
                c = (line.Get_c() * (ELEMENT - last_buffer_position))/ ELEMENT;

                Calculate_line_admittance(line.last_buffer_admittance, t_admittance, r,c);
        }
}
```

```
void Calculate_line_admittance(Admittance& head,Admittance tail, double line_R ,double line_C){

		double r,c;
		double a1,a2,a3;

		r = line_R;
		c = line_C;

		a1 = tail.y1;
		a2 = tail.y2;
//		a3 = tail.y3;

	head.y1 = a1 + c;
	head.y2 = a2 + (a1*c*r)/2. + (pow(c,2)*r)/6. + (a1 + c)*(-(a1*r) - (c*r)/2.);
/*	  head.y3 = a3 + (a2*c*r)/2. + (a1*pow(c,2)*pow(r,2))/24.0 +
			(pow(c,3)*pow(r,2))/112.0 +
			(-(a1*r) - (c*r)/2.)*(a2 + (a1*c*r)/2. + (pow(c,2)*r)/6.) +
			(a1 + c)*(-(pow(c,2)*pow(r,2))/24.0 +
			pow(a1*r + (c*r)/2.,2) - r*(a2 + (a1*c*r)/6.));
*/
		head.flag = 1;
}
```

```
/* delay.cc */

/*********************************************************/
/*********************************************************/
/*                calculate delay file                   */
/*********************************************************/
/*********************************************************/

#include "line.h"

extern double    rtr;
extern double    ctr;
extern int AWE_MAX ;


/* calculate circuit delay ( critical path ) */

void Circuit::Calculate_circuit_delay(void){
        int i;
        double delay = 0;
        double branch_delay = 0;

        for( i = 1 ; i <= total_line ; i++){
                line[i].Clear_head_flag();
                line[i].Clear_tail_flag();
        }

        for( i = 1; i <= total_line ; i++){
                if( (line[i].right == NULL ) && (line[i].left == NULL)){

                        Calculate_branch_node_delay(line[i]) ;

                        /* add delay from leaf to root */
                        branch_delay = Calculate_delay(&(line[i]));

                        /* store the worst delay */
                        if(delay < branch_delay ) delay = branch_delay;

                }
        }
        Cdelay = delay;
}

/* calculate circuit delay on specified node */

double Circuit::Calculate_specified_node_delay(Line& endline){
        int i;
        double delay = 0;

        if( (endline.right != NULL ) | | (endline.left != NULL)){
                printf("This isn't an end node! ¥n");
                exit (0);
        }
        Calculate_branch_node_delay(endline) ;
        delay = Calculate_delay(&(endline));

        return(delay);
}


/* calculate voltage on every node */

void Circuit::Calculate_branch_node_delay(Line& line){
        double delay = 0 ;
        if(line.Get_total_buffer() != 0){
                Calculate_admittance(line);

                line.Calculate_buffer_buffer_delay();

                if(line.parent != NULL){
                        Calculate_branch_node_delay(*(line.parent));
                }
                Calculate_voltage(line);

                laplace_transform(line.first_buffer_voltage , line.first_buffer_vt    ,AWE_MAX);
                line.Set_Bdelay(line, get_delay_from_vt(line.first_buffer_vt));

        }else{
                if(line.parent != NULL){
```

```
                                Calculate_branch_node_delay(*(line.parent));
                        }else{
                                Calculate_admittance(line);
                        }

                        line.Set_Bdelay(line, 0);
                        line.Set_BBdelay(line, 0);

                }

                if((line.right == NULL) && ( line.left == NULL)){
                        Calculate_voltage(line);

                        laplace_transform(line.tail_voltage , line.vt    , AWE_MAX);
                        line.Set_Ldelay(line,get_delay_from_vt(line.vt));

                }
}

/* add delay time from root to the node */

double Circuit::Calculate_delay(Line* line){
                double node_delay = 0;
                Line    *node;

                node = line;

                while(node != NULL){
                        node_delay +=    node->Get_Ldelay() ;
//                      printf("+Ldelay %e¥t",node->Get_Ldelay());
                        node_delay +=    node->Get_Bdelay() ;
//                      printf("+Bdelay %e¥t",node->Get_Bdelay());
                        node_delay +=    node->Get_BBdelay();
//                      printf("+BBdelay%e¥n",node->Get_BBdelay());
                        node = node->parent ;
                }

                return node_delay;
}


void Line::Calculate_buffer_buffer_delay(void){

                int i;
                double length;
                double line_r,line_c; /* buffer to buffer R and C */
                Admittance h_adm,t_adm;
                Voltage h_vol,t_vol;
                Vt t_vt;
                double delay ;

                delay = 0;
                for( i = 1 ; i < total_buffer; i++){
                        length = buffer[i+1].position - buffer[i].position;
                        line_r = r * length / ELEMENT;
                        line_c = c * length / ELEMENT;

                        /* calculate admittance */

                        t_adm.y1 = ::ctr * buffer[i+1].size ;

                        Calculate_line_admittance(h_adm,t_adm,line_r,line_c);

                        /* calculate voltage */

                        Calculate_root_voltage(h_vol,h_adm,buffer[i].size);
                        Calculate_line_voltage(h_vol,t_vol,t_adm,line_r,line_c);

                        /* Laplace transform and get delay */
                        laplace_transform(t_vol,t_vt,1);

                        delay += get_delay_from_vt(t_vt);

                }
                BBdelay = delay;
}
```

```
/* power.cc */

/*************************************************************/
/*************************************************************/
/*                      calculate power                      */
/*************************************************************/
/*************************************************************/

#include "line.h"

extern double Vdd;
extern double ctr;

void Circuit::Calculate_circuit_power(void){
        int i;
        double total_c = 0;

        for( i = 1; i <= total_line; i++){
                total_c += line[i].Get_c();
                if(line[i].Get_total_buffer() != 0){
                        total_c += line[i].Calculate_line_power();
                }
        }
        power = total_c * Vdd * Vdd;
}

double Line::Calculate_line_power(void){

        int i;
        double line_c = 0;

        for( i = 1; i <=    total_buffer ; i++){
                line_c += buffer[i].size * ::ctr;
        }
        return(line_c);
}
```

```
/* awe.cc */
/*******************************************************/
/*******************************************************/
/*                    AWE transform file                */
/*******************************************************/
/*******************************************************/

#include "line.h"

Vt::Vt(void){
        tau0 = 0;

        delay = 0;
        tau = 0;

        k1 = k2 = 0;
        tau_1 = tau_2 = 0;
}

void Print_vt(Vt vt){
        printf("moment = %d¥n",vt.moment);
        switch(vt.moment){
        case 0:
                printf("tau = %e¥n",vt.tau0);
                break;
        case 1:
                printf("delay = %e¥n",vt.delay);
                printf("tau   = %e¥n",vt.tau);
                break;
        case 2:
                printf("k1 = %e¥n",vt.k1);
                printf("k2 = %e¥n",vt.k2);
                printf("tau_1 = %e¥n",vt.tau_1);
                printf("tau_2 = %e¥n",vt.tau_2);
                break;
        }
}

void laplace_transform(Voltage v, Vt& vt, int moment){

        double a,b,c;

        vt.moment = moment;

        /* make positive */
        a = -1.0 * v.v0;
      b = v.v1;
//       c = -1.0 * v.v2;


        switch(moment){


        case  1:
                /* simple form - delay and tau */

                vt.delay = a - sqrt( 2.0* b - a * a);
                vt.tau = sqrt( 2* b - a * a);
                break;

        case  0:
                /* elmore model */
                vt.tau0 =  a;
                break;


        }
}
```

```
/* buffer.cc */

/*********************************************************/
/*********************************************************/
/*          file for buffer insertion and delete         */
/*********************************************************/
/*********************************************************/


#include "line.h"

Line::Line(double resistance,double capacitance,Line* ptp,Line* ptr,Line* ptl){
        r = resistance;
        c = capacitance;

            sink_c_size = 0; /* changed 99/1/11    */

            right = ptr;
            left = ptl;
            parent = ptp;

            Ldelay = 0;
            Bdelay = 0;
            BBdelay = 0;

        total_buffer = 0;
        buffer[0].position = 0;
        buffer[1].position = ELEMENT;
        buffer[0].size = 0;
        buffer[1].size = 0;
}


Line::Line(void){
            right = NULL;
            left = NULL;
            parent = NULL;

            Ldelay = 0;
            Bdelay = 0;
            BBdelay = 0;

        total_buffer = 0;
            r = 0;
            c = 0;
        buffer[0].position = 0;
        buffer[1].position = ELEMENT;
        buffer[0].size = 0;
        buffer[1].size = 0;
}


void Line::Add_buffer(void){
            double average_size;
            int i,m,n;
            int flag = 0;
            double tmp_position;
            double tmp_size;
            double probability;

        if(total_buffer == 0){
                    total_buffer++;

                    if(probability < 0.5){
                            buffer[1].position = ELEMENT * RAND;
                            buffer[1].size = (RAND * 10.0)+ 1.0 ; /* Changed */
                            buffer[2].position = ELEMENT;
                            buffer[2].size = 0;
                    }else{
                            buffer[1].position = 1;
                            buffer[1].size = M_E;
                            buffer[2].position = ELEMENT;
                            buffer[2].size = 0;
                    }

            }else if (total_buffer == MAX_BUFFER -1 ){
                    return ;
            }else{
                    tmp_position = RAND * ELEMENT ;
```

```
                        average_size = Calc_ave_size();
                        tmp_size = average_size;
                        n = ++total_buffer;

                        /* sorting */

                        flag = 0;
                        i = 0;
                        while( ( i < total_buffer) && (buffer[i].position < tmp_position) ) i++;

                        m = i;
                        for( i = total_buffer; i > m; i--){
                                buffer[i].position = buffer[i-1].position;
                                buffer[i].size = buffer[i-1].size;
                        }
                        buffer[m].position = tmp_position;
                        buffer[m].size = tmp_size;

                        buffer[total_buffer +1].position = ELEMENT;
                        buffer[total_buffer +1].size = 0;

        }
}

void Line::Del_buffer(void){
        int del_number = (int)(RAND * (double)total_buffer) + 1;
        int i;

        if( total_buffer == 0){
                return ;
        }
        for( i = del_number; i < total_buffer ; i++){
                buffer[i].position = buffer[i+1].position;
                buffer[i].size = buffer[i+1].size;
        }
        buffer[total_buffer].position = ELEMENT;
        buffer[total_buffer].size = 0;
        total_buffer--;
}

void Line::Equally_divide(int k,int h){

        int i;

        if( h < 1.0 ){
                printf("size must be positive!¥n");
                exit    (0);
        }
        total_buffer = k;

        for( i = 1; i <= k; i++){
                buffer[i].position = i * ((double)ELEMENT / (double)(k + 1.0));
                buffer[i].size = h;
        }
        buffer[k+1].position = ELEMENT;
        buffer[k+1].size = 0;
}



double Line::Calc_ave_size(void){
        int i;
        double sum = 0;
        for( i = 1 ; i <= total_buffer ; i++){
                sum += buffer[i].size;
        }
        return (sum/total_buffer);

}


void    Line::Print_buffer(void){
        int i;
        for( i = 0; i <= total_buffer + 1; i++){
                printf("buffer[%d] position = %f¥tsize = %f¥n",i,buffer[i].position,buffer[i].size);
        }
}
```

```
void Line::Print_line(void){
        printf("line r = %e¥t c=%e¥n",r,c);
}


int   Line::Make_buffer_large(int number){
        if(total_buffer == 0){
                return (-1);
        }else if(( number < 0) | | (total_buffer <number) ){
                printf(" There isn't such buffer!¥n");
                exit(0);
        }
        buffer[number].size += 1.0;
        return (1);
}

int   Line::Make_buffer_small(int number){
        if(total_buffer == 0){
                return (-1);
        }else if(( number < 0) | | (total_buffer <number) ){
                printf(" There isn't such buffer!¥n");
                exit(0);
        }

        if(buffer[number].size > 2.0){
                /* size must be larger than 1.0 */
                buffer[number].size -= 1.0;
                return (1);
        }else{
                return (0);
        }
}

int Line::Move_buffer_right(int number){
        if(total_buffer == 0){
                return (-1);
        }else if(( number < 0) | | (total_buffer <number) ){
                printf(" There isn't such buffer!¥n");
                exit(0);
        }

        if(buffer[number + 1].position - buffer[number].position > 1){
                buffer[number].position += 1.0;
                if(buffer[number].position >ELEMENT ) buffer[number].position = ELEMENT;
                return (1);
        }else{
                buffer[number].position = buffer[number + 1].position;
                return (1);
        }
/*      }else{
                return (0);
        }
*/
}

int Line::Move_buffer_left(int number){
        if(total_buffer == 0){
                return (-1);
        }else if(( number < 0) | | (total_buffer <number) ){
                printf(" There isn't such buffer!¥n");
                exit(0);
        }
        if(buffer[number].position - buffer[number - 1].position > 1){
                buffer[number].position -= 1.0;
                if(buffer[number].position < 0) buffer[number].position = 0;
                return (1);
        }else{
                buffer[number].position = buffer[number - 1].position;
                return (1);
        }

}

void Line::Clear_head_flag(void){

        head_admittance.flag                = 0;
        first_buffer_admittance.flag        = 0;
```

```
	head_voltage.flag		= 0;
	first_buffer_voltage.flag	= 0;
}

void Line::Clear_tail_flag(void){

	last_buffer_admittance.flag	= 0;
	tail_admittance.flag		= 0;

	last_buffer_voltage.flag	= 0;
	tail_voltage.flag		= 0;
}
```

```
/* bufferio.cc */

/*****************************************************************/
/*****************************************************************/
/*        insert buffer from buffernet file                      */
/*****************************************************************/
/************************************************** 98/7/2    */

#include "line.h"
FILE *BUFFERNET;
const int maxline = 1000; /* net file max line length */
const int maxlinen = 100;

/* buffer insertion from buffernet file */

void Circuit::Insertion_from_file(void){

        char buff[100];
        int linename ; /* line name */
        long position; /* buffer position */
        double size ;       /* buffer size */

    BUFFERNET = fopen("buffernet","r");

    while(fgets(buff,maxline,BUFFERNET)){
                if(!strcmp(buff,"")) continue; /* skip if blank */
            if(sscanf(buff, "buffer %d %d %le",&linename,&position,&size)){
//                      printf("buffer %d %d %le¥n",linename,position,size);
                            line[linename].Insert_buffer(position,size);
                }
        }
        fclose(BUFFERNET);
}


/* buffer insertion in line */
/* !!!! CAUTION !!!! */
/* the output of this function must be sorted when insertion ended */
/* or input must be sorted */
/* !!!! CAUTION END !!!! */

void Line::Insert_buffer(double position,double size){

        buffer[total_buffer + 1 ].position = position;
        buffer[total_buffer + 1 ].size     = size ;

        total_buffer +=    1;

        buffer[total_buffer + 1 ].position = ELEMENT;
        buffer[total_buffer + 1 ].size     = 0 ;

}

void Circuit::Output_buffer(char *buffering_file_name){

        int i,j;

        if( buffering_file_name == NULL){
                BUFFERNET = fopen("best_buffering" ,"w");
        }else{
                BUFFERNET = fopen(buffering_file_name,"w");
    }

        for( i = 1 ; i <= total_line ; i++){
                for( j = 1 ; j <= line[i].Get_total_buffer(); j++){

        fprintf(BUFFERNET,"buffer %d %e %e¥n",i,line[i].buffer[j].position,line[i].buffer[j].size);
                }
        }
        fclose(BUFFERNET);
}
```

```
/* netio.cc */

/*************************************************************/
/*************************************************************/
/*        make tree connection from net file                 */
/*************************************************************/
/************************************************** 98/6/28 */

#include "line.h"

FILE *TREENET;
const int maxline = 1000; /* net file max line length */
const int maxlinen = 100;

extern double rtr;
extern double ctr;
extern double drivingbuffer;

class NODE{
  public:
        int L1; /* parent */
        int L2;
        int L3;
        NODE(void){
                L1 = 0;
                L2 = 0;
                L3 = 0;
        }
};

void Circuit::Makecircuit(int& calline,char *net_file_name){
        int src,dest;
            int plotnode;
            int topnode;
            int endnode;

        double sink_c; /* sink capacitance size */
        double rl,cl;
        char buff[100];
        int noden;

        NODE node[maxlinen];
            int i ,j;

            if( net_file_name == NULL){
                        TREENET = fopen("net","r");
            }else{
                        TREENET = fopen(net_file_name,"r");
        }

        i = 1;        /* line numbering */
                        /* line[1] is root node */
        noden = 0;

        while(fgets(buff,maxline,TREENET)){
                        if(!strcmp(buff,"")) continue; /* skip if blank */
                if(sscanf(buff, "line %d %d %lf%lf%lf",&src,&dest,&rl,&cl,&sink_c)){
                        printf("line %d %d¥t%d¥t%e¥t%e¥t%e¥n",i, src,dest,rl,cl,sink_c);
                                line[i].Set_rc(rl,cl,sink_c);

                        /* line connectiong */

                                /* exit if tree error */
                                if( ( (node[src].L2 != 0 ) && (node[src].L3 != 0))
                                    | | (node[dest].L1 != 0)){
                                        printf("tree error !¥n");
                                        exit (0);
                                }

                                /* src node check */
                                if(node[src].L2 == 0){
                                        node[src].L2 = i;
                                }else if(node[src].L3 == 0){
                                        node[src].L3 = i;
                                }
                                /* src node connect */
                                if(node[src].L1 != 0){
                                        Connect_line(node[src].L1 ,i);
                                }
```

```
                                        /* dest node */
                                        node[dest].L1 = i;
                                        if(node[dest].L2 != 0){
                                                Connect_line(i,node[dest].L2);
                                        }
                                        if(node[dest].L3 != 0){
                                                Connect_line(i,node[dest].L3);
                                        }

                                i++;
                                        continue;
                        }
                        if(sscanf(buff, "plot %d",&plotnode)){
                                        printf("plot %d¥n", plotnode);
                                }
                                if(sscanf(buff, "treetop %d %lf %lf %lf",&topnode,&rtr,&ctr,&drivingbuffer)){
                                        printf("treetop %d %e %e %e¥n",topnode,rtr,ctr,drivingbuffer);
                                }
                                if(sscanf(buff, "treeend %d",&endnode,&ctr)){
                                        printf("treeend %d¥n",endnode,ctr);
                                }

                   }

                fclose(TREENET);

                   calline = node[plotnode].L1;
};

void Circuit::Insert_buffer(char *buffer_file_name){

                FILE *BUFFERNET;
                char buff[maxline];

                int i;
                int line_n,line_back;
                double position ,size;

                BUFFERNET = fopen(buffer_file_name,"r");
                line_back = 0;
                i = 1;

                while(fgets(buff,maxline,BUFFERNET)){
                                if(!strcmp(buff,"")) continue; /* skip if blank */
                        if(sscanf(buff, "buffer %d %lf%lf",&line_n,&position,&size)){

                                printf("%d¥t%e¥t%e¥n",line_n,position,size);

                                if( line_back != line_n){
                                        line[line_back].buffer[i].position = ELEMENT;
                                        line[line_back].buffer[i].size = 0;
                                        line[line_n].buffer[0].position = 0;
                                        line[line_n].buffer[0].size = 0;
                                        line_back = line_n;
                                        i = 1;
                                }
                                line[line_n].buffer[i].position = position;
                                line[line_n].buffer[i].size         = size;
                                line[line_n].total_buffer += 1;
                                i++;
                        }
                        line[line_n].buffer[i].position = ELEMENT;
                        line[line_n].buffer[i].size = 0;
                }
                fclose(BUFFERNET);
};
```

```
/* connection.cc */
/*****************************************************************/
/*****************************************************************/
/*                        line connection file                   */
/*****************************************************************/
/*****************************************************************/

#include "line.h"

void Circuit::Connect_line( int parent, int child){

        if( (line[parent].right != NULL) && (line[parent].left != NULL)){
                printf("Can't connect this line!!¥n");
                exit (0);
        }else if(line[parent].right == NULL){
                line[parent].right = &line[child];
                line[child].parent = &line[parent];
        }else if(line[parent].left == NULL){
                line[parent].left = &line[child];
                line[child].parent = &line[parent];
        }
        total_line++;
}

void Line::Set_rc(double resistance, double capacitance,double sink_capacitance_size){
        r = resistance;
        c = capacitance;
        sink_c_size = sink_capacitance_size;
}
```

```
/* SDfunc.cc */

/************************************************/
/************************************************/
/* functions for simulated diffusion            */
/************************************************/
/************************************************/

#include "line.h"

int Line::Move_to_min_position(int number){
        if(total_buffer == 0){
                return (-1);
        }
        buffer[number].position = buffer[number -1].position;
}

int Line::Optimum_sizing(int number,double dx){
        if(total_buffer == 0){
                return (-1);
        }
        if(1.0 < (buffer[number].size + dx) ){ /* changed on January 5 */
                                                /* size must be larger than 1 */
                buffer[number].size += dx;
//              printf("optimum sizing!¥n");
        }else{
                buffer[number].size = 1.0;
        }
        return (1);
}

int Line::Optimum_positioning(int number,double dx){
        double   min,max;
        double   current;

        if(total_buffer == 0){
                return (-1);
        }

        current = buffer[number].position ;
        min = buffer[number - 1].position ;
        max = buffer[number + 1].position ;

        if( (current + dx) < min){
                buffer[number].position = min;
        }else if( max < (current + dx)){
                buffer[number].position = max;
        }else{
                buffer[number].position += dx;
//              printf("optimum positioning! %d¥t%e¥n",number,buffer[number].position);

        }
        return (1);

}
```

```
/* SA.cc */

/*************************************************************/
/*************************************************************/
/*                  Simulated Annealing                      */
/*************************************************************/
/*************************************************************/
/* modeified    2/10 */

/*******************************/
/* input file net              */
/* output       best_buffering */
/*******************************/

/* set optimize type */
int OPTIMIZE_TYPE =    0    ; /* set "0" when delay is optimized */
                              /* set "1' when pd       is optimized */
const int BAKOGLU_IMPROVE = 0; /* set "0" if initial state is non_buffer */
                               /* set "1" if initial state is Bakoglu's */


#include "line.h"
#define WAVEOUT (0) /* set 0 in SA file */
#define PRINT_TEMPERATURE (0) /* output tempareture and delay */

Voltage input_voltage(1,0,0,0);
                    /* transistor input voltage ( step function) */

int AWE_MAX = 1; /* This parameter is set "2" in SA file. */
                 /*                       set "1" for single line */
                    /* And set by argument in awecal file */

/* simulated annealing constant parameter */
const int ONE_LOOP = 100; /* more than 10 */
const double D_EPSILON = 1e-12 ;
const double PD_EPSILON = 1e-15 * 1e-12;
const double INITIAL_TEMPERATURE_FACTOR = 0.1;
const double COOLING_FACTOR             = 0.9;

/* simulated diffusion ? */
const int DIFFUSION = 1; /* set 1 when simulated diffusion */

FILE *DTRANSITION;
FILE *SAOUT; /* OUTPUT FILE    (delay ,power ,pd ) */

/********************/
/* global variable */
/********************/

double Vdd = 1.0;
double drivingbuffer;
double ctr ; /* transistor capacitance */
double rtr ;    /* transistor resistance    */

/*************************************************/
/* simulated anneling (diffusion) parameter */
/*************************************************/

const int ADD_BUFFER        = 0;
const int DEL_BUFFER        = 1;
const int DEL_ADD_BUFFER = 2;
const int SMALL_BUFFER      = 3;
const int LARGE_BUFFER      = 4;
const int MOVE_RIGHT        = 5;
const int MOVE_LEFT         = 6;

const int SIZE_D            = 0;
const int POSITION_D        = 1;

const int TOTAL_NUMBER_OF_STATE = 7;

/***********************************/
/*          main program           */
/***********************************/

int main(int argc , char *argv[]){
            char net_file[20];
            char out_file[20];
```

```
    Circuit circuit;
    Circuit backup;
Circuit best_circuit;

    int buff;

    int line_no;
    int change_state_flag ;

    int line_total; /* total of line number */
    int flag = 0;
int diffusion = 0;
    int i, j;

    double better_delay;
    double better_power;
    double better_pd;

    double best_delay;
    double best_power;
    double best_pd;

    double delay;
    double power;
    double pd;

double delaym,delayp;
double powerm,powerp;
double pdm,pdp;

    int dummy;

    /* for simulated annealing */

    double EPSILON;
    double delta = D_EPSILON * 10.0;
    double deltapd;
    double best_delta = delta;
    int loop_no = 1;
    double temperature ; /* this value is set for first loop */
    double probability;

/* for simulated diffusion */
double f1,f2;
double current;
double dx;

int minsize,maxsize;                /* flag */
int minposition,maxposition;    /* flag */
/*****************************/

    /* decide optimize value */
    /*
    if(( argc == 2) && (strcmp("1",argv[1]) == 0))
            OPTIMIZE_TYPE = 1;
else
            OPTIMIZE_TYPE = 0;
    */
    if( argc >= 2){ /* input net file name */
                    strcpy(net_file,argv[1]);
                    printf("Input net file name - ¥"%s¥"¥n",net_file);
}else{
                    strcpy(net_file,"net");
                    printf("Input net file name - ¥"net¥"¥n");
    }
    if(( argc == 3) && (strcmp("1",argv[2]) == 0))
            OPTIMIZE_TYPE = 1;
else
            OPTIMIZE_TYPE = 0;

    /* make circuit from "net" file */

    circuit.Makecircuit(dummy,net_file);
    if(BAKOGLU_IMPROVE)             circuit.Insertion_from_file();
                                /* if BAKOGLU_IMPROVE get bakofile */

    line_total = circuit.total_line;

printf("number of lines = %d¥n",line_total);
```

```
        /*******************************/


        /* initial delay and power value */

        circuit.Calculate_circuit_delay();
        circuit.Calculate_circuit_power();

        best_circuit = circuit ;

        best_delay = better_delay = delay =    circuit.Return_d();
        best_power = better_power = power = circuit.Return_p();
        best_pd       = better_pd       = pd       = circuit.Return_pd();

        printf("power = %e¥t",power );
        printf("delay = %e¥t",delay );
        printf("pd = %e¥n",pd);
        /*********************************/


        /* flag */
        line_no = 0;
        change_state_flag = 0;
        /********/


#if ( PRINT_TEMPERATURE)
                DTRANSITION = fopen("tdelay","w"); /* transition of delay */
#endif

        if(OPTIMIZE_TYPE == 0){
                EPSILON = D_EPSILON;
        }else{
                EPSILON = PD_EPSILON;
        }

        while(best_delta > EPSILON){
                best_delta = 0;
                for( i = 0 ; i < ONE_LOOP; i++){


                    backup = circuit;       /* preserve current state */

                    /* change flags */

                            line_no = line_no % line_total;
                            line_no++;                                          /* min = 1 */

                            change_state_flag += (int) (2.0 * RAND);

                            buff = (int)(RAND * circuit.line[line_no].Get_total_buffer()) + 1;

                    if(DIFFUSION){
                             diffusion += (int)(2.0 * RAND);
                    }


                            /* if BAKOGLU IMPROVE
                                the number of buffer is constant */

//                          if( (BAKOGLU_IMPROVE) && ( i % 7 < 3 )){
//                                  continue;
//                          }

                    if(diffusion % 2 == 0 ){
                             switch(change_state_flag % TOTAL_NUMBER_OF_STATE){

                             case ADD_BUFFER:
                                     circuit.line[line_no].Add_buffer();
                                     break;
                             case DEL_BUFFER:
                                     circuit.line[line_no].Del_buffer();
                                     break;
                             case DEL_ADD_BUFFER:
                                     circuit.line[line_no].Del_buffer();
                                     circuit.line[line_no].Add_buffer();
                                     break;
                             case SMALL_BUFFER:
```

```
                circuit.line[line_no].Make_buffer_small(buff);
                break;
        case LARGE_BUFFER:
                circuit.line[line_no].Move_buffer_right(buff);
                break;
        case MOVE_RIGHT:
                circuit.line[line_no].Move_buffer_left(buff);
                break;
        case MOVE_LEFT:
                circuit.line[line_no].Make_buffer_large(buff);
                break;

        }
        circuit.Calculate_circuit_delay();
        circuit.Calculate_circuit_power();

        delay = circuit.Return_d();
        power = circuit.Return_p();
        pd      = circuit.Return_pd();

    }else if(diffusion % 2 == 1){
        switch(change_state_flag % 2){
        case SIZE_D:
                if(circuit.line[line_no].Make_buffer_small(buff) ==0){
                        minsize = 1;

                }else{
                        circuit.Calculate_circuit_delay();
                        circuit.Calculate_circuit_power();

                        delaym = circuit.Return_d();
                        powerm = circuit.Return_p();
                        pdm      = circuit.Return_pd();

                }
                circuit = backup;
                if(circuit.line[line_no].Make_buffer_large(buff) == 0){
                        maxsize = 1;
                }else{
                        circuit.Calculate_circuit_delay();
                        circuit.Calculate_circuit_power();

                        delayp = circuit.Return_d();
                        powerp = circuit.Return_p();
                        pdp      = circuit.Return_pd();
                }


        case POSITION_D:

                if(circuit.line[line_no].Move_buffer_right(buff) == 0){
                        maxposition = 1;
                }else{
                        circuit.Calculate_circuit_delay();
                        circuit.Calculate_circuit_power();

                        delayp = circuit.Return_d();
                        powerp = circuit.Return_p();
                        pdp      = circuit.Return_pd();

                }
                circuit = backup;
                if(circuit.line[line_no].Move_buffer_left(buff) == 0){
                        minposition = 1;
                }else{

                        circuit.Calculate_circuit_delay();
                        circuit.Calculate_circuit_power();

                        delaym = circuit.Return_d();
                        powerm = circuit.Return_p();
                        pdm      = circuit.Return_pd();
                }
        }

        circuit = backup;

        if( OPTIMIZE_TYPE == 0){
                current = better_delay;
```

```
                f1 = (delayp - delaym ) / 2.0;
                f2 = (delayp + delaym - 2.0 * better_delay) / 1.0;

        }else if(OPTIMIZE_TYPE == 1){
                current = better_pd;
                f1 = (pdp - pdm ) / 2.0;
                f2 = (pdp + pdm - 2.0 * better_pd) / 1.0;
        }

        if(f2 <= 0){                                    /* (d/dx)((df/dx)) < 0 */
                if(f1 > 0){                             /* df/dx > 0 */
                        dx = - 1.0;
                }else{
                        dx = 1.0;
                }
        }else{
                dx =    - ( f1/ f2);
        }

        /* exceptional case */
        if( minsize == 1){
                dx = 0;
        }else if( minposition == 1){
                if(maxposition == 1){
                        circuit = backup;
                        circuit.line[line_no].Move_to_min_position(buff);

                }else if( current > delayp){
                        dx = 1.0;
                }else{
                        dx = 0;
                }
        }else if( maxposition == 1){
                if(current > delaym){
                        dx = -1.0;
                }else{
                        dx = 0;
                }
        }
        /* exceptional case end */

        /* change to optimum value */
        if( (maxposition == 1) && ( minposition == 1)){
                /* do nothing */
        }else if( change_state_flag % 2 == SIZE_D){
                if( dx != 0) circuit.line[line_no].Optimum_sizing(buff,dx);
        }else if( change_state_flag % 2 == POSITION_D){

                if( dx != 0){
                        circuit.line[line_no].Optimum_positioning(buff,dx);
                }
        }
        circuit.Calculate_circuit_delay();
        circuit.Calculate_circuit_power();

        delay = circuit.Return_d();
        power = circuit.Return_p();
        pd      = circuit.Return_pd();

        /* clear flag */
        minsize = maxsize = 0;
        minposition = maxposition = 0;
        /***************/

}



        /* simulated annealing */
        if( OPTIMIZE_TYPE == 0){
                delta = delay - better_delay;
                deltapd = pd -    better_pd;
        }else if(OPTIMIZE_TYPE == 1){
                delta = pd - better_pd;
                deltapd = delta;
        }else{
                printf("What should be optimized ?¥n");
        }
```

```
                        if( loop_no == 1 ){
                                probability = 0;
                        }else{
                                probability = exp( - delta / temperature);
                        }

                        if( (delta < 0) | | (delta == 0 && deltapd < 0) ){          /*    adopt , of course    */
                         if( best_delta < - delta){ /* preserve delta */
                                        best_delta = - delta;
                                }

                                better_delay = delay;
                                better_power = power;
                         better_pd      = pd;


                                if( (better_delay < best_delay) && (OPTIMIZE_TYPE == 0) ){
                                        best_delay = delay;
                                        best_power = power;
                                        best_pd = pd;

                                        best_circuit = circuit;
                                }else if( (better_pd < best_pd) && (OPTIMIZE_TYPE == 1) ){
                                best_delay = delay;
                                        best_power = power;
                                        best_pd = pd;

                                        best_circuit = circuit;
                         }

                        }else if((OPTIMIZE_TYPE == 0) && (delta == 0)){             /* abondon */
                                circuit = backup;

                        }else if( RAND < probability ){    /* adopt */
                                if( best_delta < - delta){ /* preserve delta */
                                        best_delta = - delta;
                                }

                                better_delay = delay;
                                better_power = power;
                         better_pd      = pd;
                        }else{                                  /* abandon */
                                circuit = backup;
                        }

                }

                if( loop_no == 1){
                        if(OPTIMIZE_TYPE == 0){
                                temperature = best_delay * INITIAL_TEMPERATURE_FACTOR;
                        }else if(OPTIMIZE_TYPE == 1){
                                temperature = best_pd * INITIAL_TEMPERATURE_FACTOR;
                        }


                }else{
                        temperature *= COOLING_FACTOR;
                }
                loop_no++;

#if (PRINT_TEMPERATURE)
                        printf("temperature = %e¥t",temperature);
                printf("delay = %e¥t",better_delay);
                printf("power delay = %e¥n",better_pd);
                        fprintf(DTRANSITION,"%e¥t",temperature);
                fprintf(DTRANSITION,"%e¥n",better_delay);
#endif

                /* if delta is sufficiently small
                    and the better state is not the best state */

                if( ( best_delta <= EPSILON) && (OPTIMIZE_TYPE == 0)
                    && ( ( better_delay - best_delay) > EPSILON )){

                        better_delay = best_delay;
                        circuit = best_circuit;
                        printf("change state !¥n");
                        best_delta = EPSILON * 2.0;
                }else if( ( best_delta <= EPSILON) && (OPTIMIZE_TYPE == 1)
```

```
                        && ( ( better_pd - best_pd) > EPSILON    )){

                                better_pd = best_pd;
                                circuit = best_circuit;
                                printf("change state !¥n");
                                best_delta = EPSILON * 2.0;
                        }

                }

                printf("best buffering ¥n");
                printf("best power = %e ",best_power );
                printf("best delay = %e ",best_delay );
                printf("best pd       = %e¥n",best_pd );
        printf("last temperature = %e¥n",temperature);

                for( i = 1 ; i <= line_total ; i++){
                        best_circuit.line[i].Print_buffer();
        }

                printf(" one loop      = %d¥n",ONE_LOOP);
                printf(" %d loops ¥n",loop_no);

#if (PRINT_TEMPERATURE)
        fclose(DTRANSITION);
#endif
        best_circuit.Output_buffer();

                SAOUT = fopen("saout","w");
                fprintf(SAOUT,"%e¥t%e¥t%e¥n",best_delay,best_power,best_pd);
                fclose(SAOUT);

                /* for output */
                strcpy(out_file,"sa");
                strcat(net_file,out_file);
                SAOUT = fopen(net_file,"w");
                fprintf(SAOUT,"%e¥t%e¥t%e¥n",best_delay,best_power,best_pd);
                fclose(SAOUT);

}
```

```
/* file name new.cc */

/*********************************************************/
/*********************************************************/
/* this file includes buffering algorithm of my own      */
/*********************************************************/
/*                                                 1998/12/4 */

#define FIRST_BUFFER (1)
/* set 1 if first buffer's position is not 0 */

#include "line.h"
double Vdd = 3.3;
double ctr ; /* transistor capacitance */
double rtr ;    /* transistor resistance    */
double drivingbuffer;

Voltage input_voltage(1,0,0,0);
                        /* transistor input voltage ( step function) */
int AWE_MAX = 1;                /* max order of AWE */


/*****************************************************************************/
/* calculate minimum resistance upward                                       */
/* this value is used to decide the postion of first repeater in the line */
/*****************************************************************************/
double Calculate_resistance_upward(Line* line){
        if(line->rep_N == 0){
                return(line->Get_r() + ::rtr/line->optW);

        }else{
                        return(::rtr/line->optW);
        }
}



/*****************************************************************************/
/*****************************************************************************/
double Calculate_capacitance_for_line(Line* line){
        if(line->total_buffer == 0){
                        if(line->right != NULL && line->left != NULL){
                                return(Calculate_capacitance_for_line(line->right)
                                        + Calculate_capacitance_for_line(line->left)
                                        + line->Get_c() );
                        }else if(line->right != NULL){
                                return(Calculate_capacitance_for_line(line->right)
                                        + line->Get_c() );
                        }else if(line->left    != NULL){
                                return(Calculate_capacitance_for_line(line->left)
                                        + line->Get_c() );
                        }else{
                                return( line->Get_sink_c_size() * ::ctr
                                        + line->Get_c() );
                        }
        }else{
                        return(line->buffer[1].size * ::ctr ¥
                                + line->buffer[1].position * line->Get_c()/ELEMENT);
        }

}

/*****************************************************************************/
/* calculate capacitance downward */
/* this value is used to decide the postion of last buffer */
/*****************************************************************************/
double Calculate_capacitance_downward(Line* line){
        double total_capacitance = 0;
      if(line->right == NULL && line->left == NULL){
                return(line->Get_sink_c_size() * ::ctr);
        }
      if(line->right != NULL){
                        total_capacitance += Calculate_capacitance_for_line(line->right);
        }
        if(line->left != NULL){
                        total_capacitance += Calculate_capacitance_for_line(line->left);
        }
        return(total_capacitance);
}
```

```
/*************************************************************************/
/* calculation of parameters neeeded for cascaded driver paramater */
/*************************************************************************/
void Cascaded_driver(Line* line, double optW,int repN, int* driver , double* tapering_factor){
	double maxW;
	int n_driver,m_driver;

	double	rl,cl;

	double tapering_factor_m,tapering_factor_n;

	rl = line->Get_r();
	cl = line->Get_c();

	maxW = ( (cl + Calculate_capacitance_downward(line) )¥
					/ (::drivingbuffer * ::ctr * M_E)) ;

	if( (maxW > optW ) | | (repN > 0)){
		maxW = optW;
		n_driver = (int) (log( maxW ) + 1.0 );
		m_driver = n_driver - 1;

		tapering_factor_n = exp(log( maxW) / (n_driver ));
		tapering_factor_m = exp(log( maxW) / (m_driver ));

		if( (n_driver ) * tapering_factor_n - (m_driver ) * tapering_factor_m < 0){
			*tapering_factor = tapering_factor_n;
			*driver = n_driver;
		}else{
			*tapering_factor = tapering_factor_m;
			*driver = m_driver;
		}

	}else{
		n_driver = (int) (log( maxW ) + 1.0);
		m_driver = n_driver - 1;


		if( n_driver <= 0){
			*driver = 0;
		}else if(n_driver == 1){

			if(   maxW * ::ctr + (1.0/maxW - 1.0)
			    * ( cl + Calculate_capacitance_downward(line))< 0 ){
				*tapering_factor = maxW;
				*driver = n_driver;
			}else{
				*tapering_factor = 1.0;
				*driver = m_driver;
			}

		}else{
			tapering_factor_n = exp(log( maxW )/ (n_driver ));
			tapering_factor_m = exp(log( maxW )/ (m_driver ));
			if(   ((n_driver ) * tapering_factor_n - (m_driver ) * tapering_factor_m ) * ::ctr
			    + (1.0/pow(tapering_factor_n, n_driver ) - 1.0/pow(tapering_factor_m, m_driver ))
			    * ( cl + Calculate_capacitance_downward(line))< 0 ){
				*tapering_factor = tapering_factor_n;
				*driver = n_driver;
			}else{
				*tapering_factor = tapering_factor_m;
				*driver = m_driver;
			}
		}
	}

	return ;

}


/*****************************************/
/* calculate max width of cascaded driver */
/*****************************************/
double Calculate_maxW(Line* line){

	double maxW ;
```

```
        maxW = ( line->Get_c() + Calculate_capacitance_downward(line) )¥
                / (    ::ctr * M_E);
        if(maxW > line->optW){
                maxW = line->optW;
        }
        return( maxW );
}


/*****************************************************************************/
/* decide first repeater size by the number of buffers */
/*****************************************************************************/
double Decide_first_repeater_size(Line* line,double optW, int* top_buffer_insert, int* n_driver){
        double length = line->rep_N;

        double right_delay = 0;
        double left_delay = 0;

            double right_length = 0;
            double left_length = 0;

            int right; /* set 1, if this is the right line */
            double size;

        *n_driver    = 0;

        if(line->parent == NULL){
                if( length >= 2.0)
                        return(::drivingbuffer * M_E);

                Cascaded_driver(line, optW, (int) length , n_driver, &size );

                if( *n_driver == 0){
                        return( -1);
                }else{
                        return(size);
                }
        }

        right_delay = line->parent->right->optimized_delay ;
        left_delay = line->parent->left->optimized_delay ;

            right_length = line->parent->right->rep_N;
            left_length    = line->parent->left->rep_N;

            if(line->parent->right == line ){
                        right = 1;
            }else{
                        right = 0;
            }


        /* Don't insert repeater if both lines are short enough */
        if( line->parent->Get_r() ¥
             * ( Calculate_capacitance_downward(line->parent)
                -   optW * ::ctr )
                    - ::rtr * ::ctr < 0 ){

                if( right_length < 2.0 && left_length < 2.0){

                        return(-1);
                }
        }

        if( right == 1){
                        size = optW * right_delay / (right_delay + left_delay );
            }else if(right == 0){
                        size = optW * left_delay / (right_delay + left_delay );
        }


            if(size < 1.0 )
                        return(1.0);
            else if(size > optW -1.0){
                *top_buffer_insert = 0;
                        return(optW);
            }else{
                        return(size);
            }
```

```
        exit(0);
}



/*************************************************************/
/*          original algorithm to find optimum buffering         */
/*************************************************************/
/* if output == -1 , this means buffer should be reduced */
/*                  == 0    , exitted normaly                                  */


int Optimize_buffering(Line* line,int division ,int trial_times){
        /* if parent_buffer = 1,
             then parent length is long enough */

        double optW;
        double optL;
        double W;
        double L;
        int repN; /* total number of repeaters */

        double rl,cl;
        double rlb;
        int total_buffer;

        int i,j,k;
        int m;
        int n; /* the number of the same length section */
        double length1;
        double length2;
        double length3;

        double parent_buffer= -1; /* this means thant this line has no parent line */
        double right_buffer = -1; /*                                                  right            */
        double left_buffer    = -1; /*                                                  left             */

        int tail_buffer_insert = 0;/* if this value is 1, tail buffer is needed */
           int top_buffer_insert = 1; /* if this value is 1, top buffer is needed */

        double capacitance_downward;
        double resistance_upward;
        double first_repeater_size;

        int n_driver ; /* the number of cascaded driver */

        double maxW ; /* maximum size of cascaded driver */

        rl = line->Get_r();
        cl = line->Get_c();

        optW = line->optW;
        optL = line->optL;

        /* input buffer[0] */
        i = 1;
        line->buffer[0].position = 0;
        line->buffer[0].size = 0;
        W = ::drivingbuffer;

        /* input some data */
        if(line->parent!= NULL) parent_buffer = line->parent->rep_N;
        if(line->right != NULL) right_buffer    = line->right->rep_N;
        if(line->left    != NULL) left_buffer      = line->left->rep_N;


           /*****************************/
        /* insert cascaded driver */
           /*****************************/
           j = 0;                          /* j is the number of the cascaded driver */

           first_repeater_size = Decide_first_repeater_size(line,optW,&top_buffer_insert, &n_driver);
        maxW = Calculate_maxW(line);

           printf("first repeater size = %e¥n",first_repeater_size);
        printf("maximum driver size = %e¥n",maxW);
```

```
        if( (first_repeater_size < optW ) && (first_repeater_size > 0) ){
                W = first_repeater_size;

                printf("max_W = %e¥n",optW);

                if(division == 0){
                        for( j = 1 ; j <= n_driver ; j++){
                                line->buffer[j].position = 0;
                                line->buffer[j].size = W;
                                W *= first_repeater_size;
                        }
                        W /= first_repeater_size;
                }else{
                        for( j= 1; W    < optW ;j++){
                                line->buffer[j].position = 0;
                                line->buffer[j].size = W    ;
                                W = W * M_E;
                        }
                        W /= M_E;
                }

                j--;


        }

            /**********************/
        /* insert repeaters */
            /**********************/
            k = 1;
        n = division - 1;    /* n is the number of the same length section */

        if( (division < 1) | | (first_repeater_size < 0) ){
                if( j != 0){
                        /* this line is root */
                        /* Or too short to insert buffer */
                    total_buffer = line->total_buffer = j ;
                                line->buffer[j + 1].position = ELEMENT;
                                line->buffer[j + 1].size = 0;
                    }else if(first_repeater_size < 0){
                                /* insert no buffer */
                        total_buffer = line->total_buffer = 0;

                        line->buffer[1].position = ELEMENT;
                        line->buffer[1].size = 0;

                }else{

                                /* insert 1 buffer */
                        total_buffer = line->total_buffer = 1;

                        line->buffer[1].position = 0;
                        line->buffer[1].size = first_repeater_size;

                        line->buffer[2].position = ELEMENT;
                        line->buffer[2].size = 0;
                    }

                printf("This line is too short to insert buffer! %d¥n",j);
                return(1);
                    /* return normally */
            }

            if( (j == 0)
#if FIRST_BUFFER
                && (top_buffer_insert == 1)
#endif
                ){
                        /* if cascaded driver is not inserted */
                        line->buffer[1].position = 0;
                        W = line->buffer[1].size = first_repeater_size;
                        j = 1;
            }
        if( j != 0)
                first_repeater_size = W ;

        /* calculate capacitance downward */
            capacitance_downward = Calculate_capacitance_downward(line);
```

```
        /* calculate resistance upward */
            if( j == 0){
                        resistance_upward = Calculate_resistance_upward(line);
            }else{
                    resistance_upward = ::rtr / W;
            }


#if FIRST_BUFFER
                length1 = ELEMENT * ( 1.0 / (n + 2.0) ¥
                                        - (resistance_upward/rl) *    (n + 1.0) / (n + 2.0) ¥
                                        + (::rtr/rl) * ((n + 1.0)/(n+2.0))/optW ¥
                                        - optW * (ctr / cl) / (n + 2.0)    ¥
                                        + (capacitance_downward /cl) / (n + 2.0));
                length2 = ELEMENT * ( 1.0 / (n + 2.0) ¥
                                        + (resistance_upward/rl) /(n + 2.0) ¥
                                        -    (::rtr/rl) * ((1.0)/(n+2.0))/optW ¥
                                        + optW * (ctr / cl) * (n +1.0) / (n + 2.0)¥
                                        - (capacitance_downward /cl) * (n + 1.0) / (n + 2.0));
#elif

                length1 = ELEMENT * ( 1.0 / (n + 2.0) ¥
                                        +    (capacitance_downward /cl) / (n + 2.0) ¥
                                        - (::rtr / rl)    * ( optW/first_repeater_size    -1.0 )* ((n + 1.0)/(n+2.0))/optW ¥
                                        - optW * (::ctr / cl) / (n + 2.0) )    ;
                length2 = ELEMENT * ( 1.0 / (n + 2.0) - (capacitance_downward /cl) * (n + 1.0) /(n + 2.0) ¥
                                        +    (::rtr/rl)      * (optW/first_repeater_size -1.0 )* ((1.0)/(n+2.0))/optW ¥
                                        + optW * (::ctr / cl) * (n +1.0) / (n + 2.0)) ;
#endif


                if( n != 0){
                            length3 = (ELEMENT - length1 - length2) / n;
                }else{
                            length3 = 0;
                }

                        /*changed section ends */

        /* if buffers are too many, the number of buffers is reduced and tried again */

        /* format error check */
        if(( length1 < 0) && (length2 < 0)){
                    length1 = 0;
                    length2 = 0;
                    tail_buffer_insert = 1;
        }else if(length1 < 0){
                    length1 = 0;
                    length2 = ELEMENT * ( 1.0 / (n + 2.0) - (capacitance_downward /cl) * (n + 1.0) /(n + 2.0) ¥
                                            + optW * (::ctr / cl) * (n +1.0) / (n + 2.0)) ;
        }else if(length2 < 0){
                    length1 = ELEMENT * ( 1.0 / (n + 2.0) ¥
                                            - (::rtr / rl)    * ( optW/first_repeater_size    -1.0 )* ((n + 1.0)/(n+2.0))/optW);
                    length2 = 0;
                    tail_buffer_insert = 1;
        }

        if(length1 < 0){
                    length1 = 0;
        }
        if(length2 < 0){
                    length2 = 0;
                    tail_buffer_insert = 1;
        }

        if( n != 0){
                    length3 = (ELEMENT - length1 - length2) / n;
        }else{
                    length3 = ELEMENT;
        }

        if( length3 < (optL*ELEMENT) ){
                    printf("length1    =    %e    length2    =    %e    length3    =    %e    OptL
= %e¥n",length1,length2,length3,optL*ELEMENT);
                    return(-1);
        }

            /* first repeater */
            line->buffer[j+k].position = length1;
```

```
		line->buffer[j+k].size = optW;

		/* insert the same size repeaters */
	for( k++; k <= n + 1 ; k++){
		line->buffer[j+k].position = line->buffer[j+k-1].position + length3;
		line->buffer[j+k].size = optW;
	}

	if(tail_buffer_insert == 1){
		/* As child line has no buffer, */
		/*	tail buffer is inserted */

		line->buffer[j+k].position = ELEMENT;
		line->buffer[j+k].size = optW;
		k++;
	}

	k--;
	total_buffer = line->total_buffer = j + k;

	printf("length1 = %e¥tlength2 = %e¥tlength3 = %e¥t division = %d¥n",length1,length2,length3,division);
	printf("j = %d¥tk =%d¥ttotal_buffer =%d¥n",j,k,line->total_buffer);

		/* for correct format */
	line->buffer[total_buffer + 1].position = ELEMENT;
	line->buffer[total_buffer + 1].size = 0;

	printf("Opt L = %e¥n",optL);
	printf("Opt W = %e¥n",optW);

	return(0);
}


/*******************************************************/
/* circuit optimization function                       */
/*                      ( this function calls itself)  */
/*******************************************************/
int Optimize_circuit(Line* line ){
	double rl,cl;
	int rep_N;
	rl = line->Get_r();
	cl = line->Get_c();
	if(line->rep_N < 1.41421356)
		rep_N = 0;
	else{
		rep_N = (int) line->rep_N;
	}

		/* insert from shorter line */
		if(line->right != NULL && line->left != NULL){
			if(line->right->optimized_delay > line->left->optimized_delay){
				Optimize_circuit(line->left);
				Optimize_circuit(line->right);
			}else{
				Optimize_circuit(line->right);
				Optimize_circuit(line->left);
			}
		}else if(line->right != NULL){
			Optimize_circuit(line->right);
	}else if(line->left    != NULL){
			Optimize_circuit(line->left);
	}


		int trial_times = 1;
		while(Optimize_buffering( line, rep_N ,trial_times) == -1){
			rep_N = rep_N -1;
			trial_times += 1;
			printf("Reduce buffer to %d!¥n",rep_N);
		}

		printf("Buffer insertion ended for one line %e!¥n¥n",line->Get_r());

}

/*****************************************************************************/
/* roughly estimate the number of buffers in each line           */
/*****************************************************************************/
```

```
void Rough_estimate_line(Line* line){
	double rl,cl;
	double rep_N;

	    double right_delay = 0;
	    double left_delay = 0;

	rl = line->Get_r();
	cl = line->Get_c();

	rep_N =    (1.0/ sqrt( 2.0 * rtr * ctr / ( rl * cl )) );

	line->optW = sqrt( ::rtr * cl / ( rl * ::ctr));
	line->optL = sqrt( 2.0 * ::rtr * ::ctr / ( rl * cl ));

	line->rep_N = rep_N;

	    line->optimized_delay = sqrt(rl * cl);
	    if( line->right != NULL){
	                right_delay = line->right->optimized_delay;
	    }
	if( line->left != NULL){
	        left_delay = line->left->optimized_delay;
	}

	    if(right_delay < left_delay){
	        line->optimized_delay += left_delay;
	}else{
	        line->optimized_delay += right_delay;
	}

	printf("rep_N =%e¥t%d¥n",rep_N,(int)rep_N);
	printf("optimized_delay = %e¥n",line->optimized_delay);

}


/*****************************************************************************/
/* roughly estimate the number of buffers                                    */
/* this function is called before Optimized_circuit function */
/*                                                  to calculate rep_N */
/*****************************************************************************/
void Rough_estimate(Line* line){

	if( line->right != NULL){
	        Rough_estimate(line->right);
	}
	if( line->left != NULL){
	        Rough_estimate(line->left);
	}
	Rough_estimate_line(line);
}

/*************************************************/
/*************************************************/
double new_algorithm(Circuit& circuit, int linen ){
	double optW;
	double optL;

	double rl,cl;
	int total_buffer;
	double rep_N; /* the number of the same length section */
	int i;
	double delay_1, delay_2, delay_3;
	    double worst_delay;
	double lmin[LINE_MAX];
	int length_check; /* set 1 if a line is long enough to insert repeaters */

	/*************************************************/
	Rough_estimate( &(circuit.line[1]) );
	    /*************************************************/
	Optimize_circuit( &(circuit.line[1]) );
	/*************************************************/

	/* delay calculation */
	delay_1 = circuit.Calculate_specified_node_delay(circuit.line[linen]);
	    circuit.Calculate_circuit_delay();
	worst_delay     = circuit.Return_d();
	/* printf buffering and delay */
```

```
	for( i = 1 ; i <=    circuit.total_line ; i++){
		circuit.line[i].Print_buffer();
	}

	printf("total_buffer = %d¥n",circuit.line[1].total_buffer);
	printf("New algorithm delay 1 = %e¥n",delay_1 );
	printf("New algorithm delay 2 = %e¥n",worst_delay);
	printf("check line = %d¥n",linen);

	    return(worst_delay);

}

/***********************************/
/* Bakoglu optimization for line */
/***********************************/
void Optimize_Bakoglu_line(Line* line){
	int j,k;
	double optW,optL;
	double W,maxW;
	int repN;
	int total_buffer;
	double rl,cl;

	double parent_buffer = -1;
	double right_buffer = -1;
	double left_buffer = -1;

	    double tapering_factor_n,tapering_factor_m;
	    double tapering_factor;
	int n_driver ,m_driver;
	int driver;

	int n_repeater,m_repeater;

	rl = line->Get_r();
	cl = line->Get_c();

	    optW = sqrt( ::rtr * cl / ( rl * ::ctr));
	optL = sqrt( 2.0 * ::rtr * ::ctr / ( rl * cl ));

	printf("optW = %e¥n",optW);
	printf("optL = %e¥n",optL);

	double input_size = 1.0/optW;
	double output_size = 1.0/optW;

	/* input buffer[0] */
	line->buffer[0].position = 0;
	line->buffer[0].size = 0;

	W = ::drivingbuffer;

	if(line->parent != NULL) parent_buffer = line->parent->rep_N;
	if(line->right != NULL ) right_buffer = line->right->rep_N;
	if(line->left !=    NULL ) left_buffer    = line->left->rep_N;

	/* decide the number of repeaters */
	n_repeater = (int ) (1.0 +    1.0/optL);
	m_repeater = n_repeater - 1;

	if(m_repeater == 0){
		repN = 0;
	}else if( ::ctr * ::rtr - rl * cl /(2.0 * n_repeater * (n_repeater - 1) ) < 0){
		repN = n_repeater;
	}else{
		repN = m_repeater;
	}
	printf("number of repeaters = %d¥n",repN);

	/* cascaded driver */

	Cascaded_driver(line,optW, repN, &driver, &tapering_factor);


	if(line->parent == NULL){
		for(j = 1; j <= driver ; j++){

			W *= tapering_factor;
```

```
                        line->buffer[j].position = 0;
                        line->buffer[j].size = W ;
                }
                j--;


        }else{
                if(line->Get_c() + ctr < optW * ctr ){
                        return;
                }

                line->buffer[1].position = 0;
                line->buffer[1].size = optW;
                j = 1;
        }


        /* division in the same length section */


        for( k = 1; k <= repN -1; k++){
                line->buffer[j+k].position = ( (double)ELEMENT / repN) * k ;
                line->buffer[j+k].size = optW;
        }
        line->buffer[j+k].position = ELEMENT;
        line->buffer[j+k].size = 0;
        k--;

        line->total_buffer = j + k;
}

/*************************************************************************/
/* Bakoglu optimization for circuit */
/*************************************************************************/
void Optimize_Bakoglu(Line* line ){

        if(line->right != NULL){
                Optimize_Bakoglu(line->right);
        }
        if(line->left != NULL){
                Optimize_Bakoglu(line->left);
        }
        Optimize_Bakoglu_line(line);
}

/*************************************************/
/*************************************************/
double Bakoglu(Circuit& circuit, int linen ){

        double optW;
        double optL;
        double W;
        double L;
        int repN; /* total number of repeaters */

        double rl,cl;
        int total_buffer;

        int i,j,k;
        int n; /* the number of the same length section */

        double delay;
           double worst_delay;

        Optimize_Bakoglu(&circuit.line[1]);

        for( i = 1 ; i <=    circuit.total_line ; i++){
                circuit.line[i].Print_buffer();
        }

        delay = circuit.Calculate_specified_node_delay(circuit.line[linen]);
        circuit.Calculate_circuit_delay();
           worst_delay = circuit.Return_d();
        printf("Bakoglu = %e¥n",delay);
        printf("Bakoglu = %e¥n",worst_delay);

        return(worst_delay);
}
```

```
/*****************************/
/*****************************/
/*        main program                */
/*****************************/
/*****************************/
int main( int argc,char *argv[]){

        Circuit circuit_1;
        Circuit circuit_2;
        Circuit circuit_3;

            double newdelay;
            double bakogludelay;

            double newpd;
            double bakoglupd;

            double newp;
            double bakoglup;

        FILE *NEW,*BAKOGLU;
            char net_file[20];
            char bako_buffer_file[20];
            char new_buffer_file[20];

            int    linen;
            double delay_1,delay_2,delay_3;
            double slope;
            int i;

            if( argc != 2){
                        strcpy(net_file,"net");
                    printf("Input net file name - ¥"net¥"¥n");
        }else{
                        strcpy(net_file,argv[1]);
                    printf("Input net file name - ¥"%s¥"¥n",net_file);
        }

        /*    input circuit file */

            circuit_1.Makecircuit(linen,net_file);
        circuit_2.Makecircuit(linen,net_file);

            /* buffer insertion */
            /* calculation */
            newdelay = new_algorithm(circuit_1, linen);
            bakogludelay = Bakoglu(circuit_2, linen);

            circuit_1.Calculate_circuit_power();
            circuit_2.Calculate_circuit_power();

            newpd = circuit_1.Return_pd();
            bakoglupd = circuit_2.Return_pd();

            newp = circuit_1.Return_p();
            bakoglup = circuit_2.Return_p();

            printf("new algorithm pd = %e bakoglu pd = %e¥n",newpd,bakoglupd);
            printf("new algorithm p = %e bakoglu p = %e¥n",newp,bakoglup);

        NEW = fopen("new.out","w");
        BAKOGLU = fopen("bakoglu.out","w");

            fprintf(NEW,"%e¥t%e¥t%e¥n",newdelay,newpd,newp);
        fprintf(BAKOGLU,"%e¥t%e¥t%e¥n",bakogludelay,bakoglupd,bakoglup);

            strcpy(new_buffer_file, "new_buffering");
            circuit_1.Output_buffer(new_buffer_file);

            strcpy(bako_buffer_file,"bako_buffering");
            circuit_2.Output_buffer(bako_buffer_file);

        fclose(NEW);
        fclose(BAKOGLU);

}
```

参考文献

[1] “The national Technology Roadmap for Semiconductor 1997,” SIA, Semantech, Inc., 1997.

[2] H. B. Bankoglu, “Circuits, Interconnections, and Packaging for VLSI”, Addison-Wesley Publishing Company, pp.211-219.

[3] H. B. Bakoglu, “Optimal Interconnection circuits for VLSI,” IEEE Transactions on Electron Devices, Vol. ED-32, No.5, pp. 903-909, May 1985.

[4] Lawrence T. Pillage, “Asymptotic Waveform Emulation for Timing Analysis” , IEEE Trans. On Computer Aided Design, Vol. 9, pp.352-366, April 1990.

[5] Lawrence Pileggi, “Coping with RC(L) Interconnect Design Headaches”, ICCAD Digest of Technical Papers, pp.246-253. Nov 5-9, 1995.

[6] S. Kirkparick, C. D. Gellatt and M. P. Vecchi, “Optimization by simulated Annealing”, Science, Vol. 220, No.4598, pp671-680, May 1983.

[7] T. Sakurai, Bill Lin, and A. Richard Newton, ”Fast Simulated Diffusion : An Optimization Algorithm for Multiminimum Problem and Its Application to MOSFET Model Parameter Extraction”, IEEE Trans. Computer-Aided Design, Vol. 11, No 2, pp228-234, Feb. 1992.

[8] Seok-Yoon Kim, Nanda Gopal, Lawrence T. Pillage, “Time Domain Macromodels for VLSI Interconnect Analysis”, IEEE Trans. On Computer-Aided Design, pp. 1257-1270, October 1994.

[9] V. Adler and E.G. Friedman, “Delay and Power Expressions for a CMOS Inverter Driving a Resistive-Capacitive Load”, Analog Integrated Circuits for Signal Processing, Vol. 14, No. 1/2, pp. 29-40, September 1997.

[10] V. Adler and E.G. Friedman, “Repeater Design to Reduce Delay and Power in Resistive Interconnect,” IEEE International Symposium on Circuits and Systems, pp. 2148-2151, June 9-12, 1997, Hong Kong.

[11] V. Raghavan, R.A.Rohrer, L.T. Pillage, J.Y.Lee, J.E.Bracken, M.M.Alaybeyi, “AWE-Inspired”, IEEE. 1993 Custom Integrated Circuits Conference.

[12] Seok-Yoon Kim, Nanda Gopal, Lawrence T. Pillage, “Time Domain Macromodels for VLSI Interconnect Analysis”, IEEE Trans. On Computer-Aided Design, pp. 1257-1270, October 1994.

[13] Chung-Yu, Wing-Chuen Shiau, “Delay Models and Speed Improvement Techniques

for RC Tree Interconnections Among Small-Geometry CMOS Inverters," IEEE Journal of Solid-state Circuits, Vol. 25, No. 5, pp.1247-1256, October 1990.

[14] M. Nekli and Y. Savaria, "Optimal Method of Driving Interconncetions in VLSI circuits," Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 21-23, May 1992.

[15] 山越 公洋, 沢田 博俊, 首藤 哲樹,「PWL によるゲート遅延解析手法」 電子情報通信学会総合大会 pp. 107 A－3－2 1997 年.

本研究に関する発表

1.「配線遅延近似精度のモーメントマッチング次数依存性」井高 康仁 桜井貴康 1998 年秋季第 59 回応用物理学会学術講演会 15p－P9 pp781

2.「ディープサブミクロン配線のリピータ挿入最適化」井高康仁 桜井貴康 1998 年 電子情報通信学会ソサイエティ大会 C-12-2 pp93

本論文以外の発表

[1] H. Kawaguchi and Y. Itaka and T. Sakurai, "Dynamic Leakage Cut-off Scheme for Low-Voltage SRAM's", Symposium on VLSI Circuits, pp. 140-141, June 1998.

[2] 川口博、 井高康仁、 桜井 貴康「低電圧SRAMのための Dynamic Leakage Cut-off 設計法」 電子情報通信学会技術研究報告、vol. 98, No. 119, 1998 年 3 月.

謝辞

本研究をここにまとめるにあたり、2 年間御指導を賜わりました指導教官の桜井貴康教授に深く感謝いたします。様々な話題の中で数多くの感銘を受け、エンジニアとして、研究者としての態度を学ぶことができました。

同じく御指導を賜りました川口技官に深く感謝いたします。

また、本研究にあたり環境を整備していただいた東芝（株）の皆様に感謝いたします。

修士課程 2 年の野瀬君をはじめ研究室の学生の皆様、秘書の山岡さんにはお世話になりました。この場をかりてお礼申し上げます。

内容梗概

本研究の目的は、微細化されたLSIにおけるツリー構造配線でのリピータ挿入の最適化についての指針を得ることである。

LSIの微細化に伴い配線も微細化され、配線の抵抗・容量の積による配線遅延の増大が問題となってきている。スケーリングによってゲート遅延は減少する一方、配線遅延が増加するため回路全体の高速化が難しくなり、配線遅延増加のための対策が必要となる。

リピータ挿入とは、配線長とともに2乗で増加する配線遅延を緩和する方法である。長い配線を間にリピータを挿入することにより分割し、配線遅延の増加を線形程度の増加に抑えることが目的である。リピータ挿入によって最適化される遅延時間は、トランジスタ・配線のパラメータで決まり、ゲート遅延が減少されればリピータ挿入の効果は増大する。

本研究では、まずリピータ挿入のための遅延時間計算モデルを考え、次に遅延時間最適化のためのリピータ挿入の方法についての研究を行った。リピータ挿入の方法として、等間隔リピータ挿入法、simulated diffusion による方法、新たに提案する方法の3つの方法について、評価を行い、結果を SPICE で検証した。

また、リピータの挿入とともに増加する消費電力も考え、低消費電力設計に向けたリピータ挿入の問題について、単一配線の場合での研究を行った。遅延時間最適設計と共に消費電力・遅延時間積（ＰＤ積）最適化設計のためのリピータ数、リピータサイズについての考察を行った。

目次