Memoirs of the Faculty of Science
Kochi University (Information Science)
Vol.29 (2008), No.8

二分木と三分木のクロックツリーの回路規模に関する研究

A Study of Scales of Binary and Triplet Clock Tree Circuits

下川 遥香　　豊永 昌彦
Haruka Shimokawa　　Masahiko Toyonaga
高知大学理学部数理情報科学科
Faculty of Science, Kochi University

あらまし

高度情報化社会の推進によって，コア LSI は，大規模化と高速化を要求されている．しかし LSI のクロック周波数の高まりは消費電力を増大させてしまうため，今後の LSI 設計ではより小規模なクロック回路の生成が重要となる．

クロック回路規模は配線と素子の総和で決まる．従って回路規模を縮小するには配線長・分木点数・段数を削減しなければならい．

著者は本論文においてクロック分木数を増やして従来の二分木と二分木以上の分木数をもつクロック回路の規模の面からの優位性について定量的な比較をおこなった．端子をランダムに配した簡単なベンチマークデータで評価したところ，三分木が二分木に比べて規模を約 7%削減できた．三分木クロック回路により総量が減ると安価・小型化が期待できる．

1　はじめに

近年の情報化の進歩に伴い，電子機器で使われるコア LSI の大規模化と高速化が市場から要求されている．

これらの LSI の多くは演算速度をクロック信号で制御する同期式回路設計方式が採用されている．同期式回路設計は，クロック信号を全フリップフロップに遅延差なく供給するクロック回路を前提としている．

一方，同期式回路ではクロック回路はほぼ常時動作し続ける．そのため，クロック回路で消費される電力も多く，LSI 全体の半分を占めているといわれている．

ところでクロック回路の設計で，遅延差無く各フリップフロップに信号を供給するためには配線長の均等化が必須でなる．そのため，従来のクロック設計法では，信号遅延の調整を容易にするため，２つのフリップフロップ端子毎に均等配線を構成する二分木のクロック構造が使われてきた．ここで分木数とは，端子間の配線長を均等化して同時に複数へ信号を送るための端子のまとまりの数である．

著者は，クロック回路規模の削減方法の検討に際して，これらの分木数に着目する．より多数の分木数による高次の木構造が有利と思われるためである．

そこで本論文では，まずクロック分木数について二分木と二分木以上の分木の回路量を比較してみた．　その結果，二分木より三分木が配線長と段数で最大 20%程度削減できることがわかった．すなわち，クロック回路の消費電力の全回路に占め割合を 50%としたとき，全消費電力 10%を削減することに相当する．三分木によるクロック回路は，低消費電力を目指す LSI において有力であることがわ

かった.

以下本論文の構成は, 2 章でクロックについて分木数と段数の関係について説明をおこなう. 3 章は, 2 分木, 3 分木について生成アルゴリズムの説明を行う. 4 章では, 2 分木と 3 分木のクロックの配線長, 段数の比較を 100x100～400x400 の配線領域に端子 50～400 個ランダムに配置した回路を作成して実験をおこなった結果と考察を紹介する. 5 章で, 本論文をまとめる.

2　クロック構造

2.1 クロックとは

クロックは信号処理を正しくおこなうためにタイミングを合わせる周期的な信号である.

LSI では, 演算回路の出力は, いったんフリップフロップに記憶され, クロック信号のタイミングで次の演算入力へと送られる. これにより, 処理結果を正しく次の処理へ伝えることができる. この回路方式を同期設計方式と呼び, デジタル信号処理の回路で主流となっている.

クロック回路の構成として**クロック･グリッド網方式**がある. これは, あらかじめチップ全体に広げたグリッド網配線を巨大なトランジスタで強引に同時駆動させる方法である. しかし, その配線規模や駆動するトランジスタが巨大となる為, 消費電力は多大となってしまう. [4]

一般的なクロック回路は, 信号源を根として, 根から葉（フリップフロップ端子）まで木構造（クロック木）により構成する**ゼロスキュークロックツリー方式**が使われている. 信号の遅延は配線長の遅れと素子の遅れで決まるので, クロック信号を全フリップフロップ端子まで同時に伝えるため, クロック木の根から各葉までの段数, 枝の長さはすべて均等化した回路がつくるものである. クロック木の分木点の素子としてはバッファ回路(信号を強める回路）が用いられる. なお, スキューとはクロック到達時間の差のことを指す. グリッド網方式より消費電力は少なくなるが, 回路規模が増大すると段数が増え, 調整する配線長も巨大化してしまう。[4]

本論文では, ゼロスキュークロック方式, すなわち分木により信号遅延を均等化する回路についてその削減方法を検討する.

2.2 n分木のクロック構造

枝をｎ本ずつとりまとめて構成した木をｎ分木という.

ｎ分木を作るには, ｎ個の端子をグループ化して分木点で取りまとめ, 再度分木点を端子とみなしてｎ個の端子をくり返しグループ化して作ることができる.

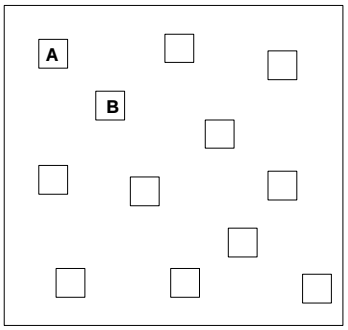
図を使って, ｎ＝２の場合の木構成の手順を説明する. 図 1(a)は, ランダムに置かれた 12 個のフリップフロップの端子を□で示したものである.

まず, 最近接している端子 n=２を図 1(a)の A および B のペアとすると, これらを１組としてグループにまとめたものを C とする(図 1(b)参照).

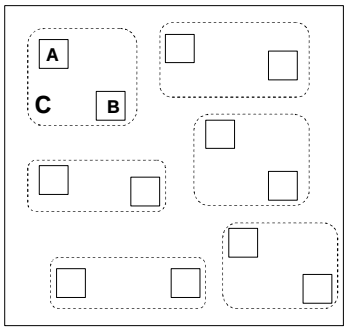
上位の端子 C は, A, B から均等な位置に置く. このようにして, 下位の全端子から最隣接するｎ個のグループをつくり, 各グループに上位端子を生成しておく.

上位端子群についても, 同様に, ｎ個の最隣接端子をグループにしてさらに上位の端子を構成していく. これを繰り返すことにより, もし上位の端子がたかだか 1 つになったなら, クロック木の構成を出力して処理は終了する.

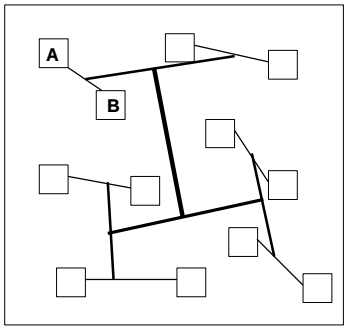
以上のようにして得られたｎ分木構造について，各グループ内の端子を直線で結んだものを図 1(c)に示す．ｎ＝２で端子総数が 12 個である本例では，分木点数が 10 点となっている．



(a)端子群



(b)ペアの生成



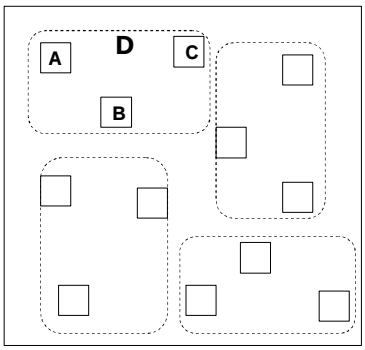
(c)二分木構造

図 1．二分木のクロック生成例

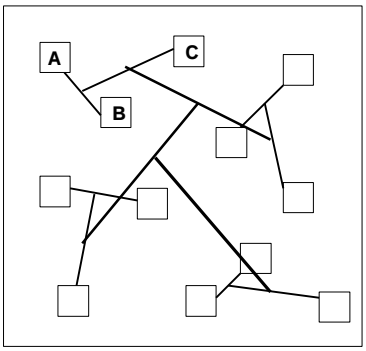
参考のためｎ＝３の三分木構造についても同様に図で示しておく，最近接となる３端子 A,B,C のグループ D を作り（図 2(a)）．次にグループ D においてA，B，C から均等な位置に上位で扱うグループ D の端子を１つ作る．

上位の端子について先ほどのグループ生成を繰り返すことで上位の端子がたかだか１つになったところで，クロック木を出力して処理は終了する．

三分木構造では，分木点数が 6 点となっている．



(a)　三端子グループの生成



(b)　三分木構造

図 2．三分木のクロック生成例

このように，クロック木の分木点の数はｎによって変わる．すなわち，

(1) より小さなｎのｎ分木では，分木点の数（素子数）が多くなる．（先ほどの例では端子数 12 個について，二分木が 10 点，三分木が 6 点である）

(2)ｎ分木で段数が少ないほど配線長が短い．（先ほどの端子数 12 個で二分木では 4 段，三分木では 3 段であるが，各段を結ぶ配線数そのものが減っている）

2.3 分木数ｎと分木点数の関係

以上の例図で見られた傾向について, 分木数**n**を使って計算により見積もることができる.

いま分木数を**n**, 全フリップフロップ端子数を N とすると，分木点数 S は,

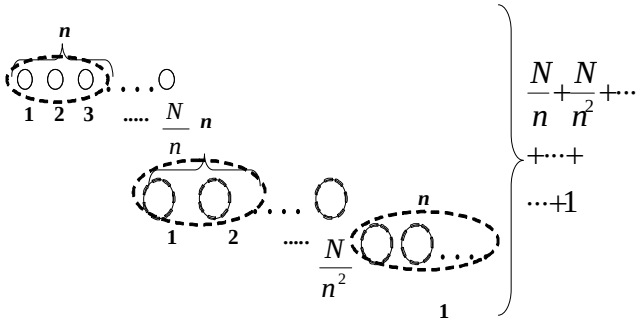


図 3. 分木点数生成

図 3 のように$\frac{N}{n},\frac{N}{n^2}$...と 1 になるまで分木点数は生成される.

式に表すと,

$$S=\frac{N}{n}\left\{1+\left(\frac{1}{n}\right)^1+\left(\frac{1}{n^2}\right)^2+\ldots+\left(\frac{1}{n}\right)^{\log_n N-1}\right\}$$

$$=N\left\{\left(\frac{1}{n}\right)^1+\left(\frac{1}{n^2}\right)^2+\ldots+\left(\frac{1}{n}\right)^{\log_n N}\right\}$$

$$=N\frac{\frac{1}{n}-\left(\frac{1}{n}\right)^{\log_n N+1}}{1-\frac{1}{n}}$$

$$=\frac{N}{n}\left(\frac{1-\left(\frac{1}{n}\right)^{\log_n N}}{1-\frac{1}{n}}\right)$$

となり, 実際に数値を入れて計算すると, 分木数を 2～5 とし，端子数 N を 100 設定すると以下の結果となる.

・$n=2$ のとき　　S　99

・$n=3$ のとき　　S　49

・$n=4$ のとき　　S　33

・$n=5$ のとき　　S　25

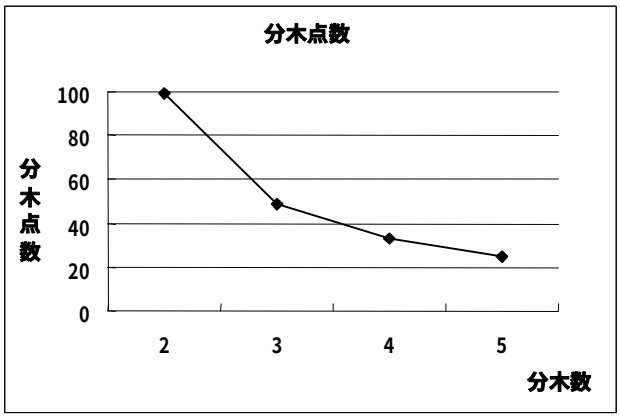


図 4. 各分岐数に対する分木点数の値

図 4 から計算上二分木から三分木にすることで, 分木点数 99 個が 49 個まで分木点数が 50%程度削減される可能性が期待できることができる．一方, それ以上の分木数の改善効果は，32%以下である.

2.4 分木数**n**と段数の関係

nと段数の関係も計算により見積もることができる．分木数を**n**とし，端子数 N とすると，**n**個ずつの端子グループを生成するので,

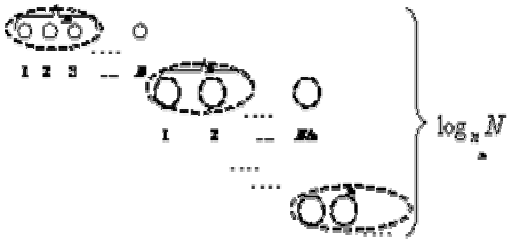


図 5. 端子グループ生成

図 5 より段数は$\log_n N$ となり，分木数**n**が大きいほど段数が削減されていることは, 実際に数値を入れて計算して確かめられる．すなわち，分木数を 2～5 とし, 端子数 N を 100 設定すると以下の結果となる.

・$\log_2 100$　6.7

・$\log_3 100$　4.0

・$\log_4 100$　3.3

・$\log_5 100$　2.9

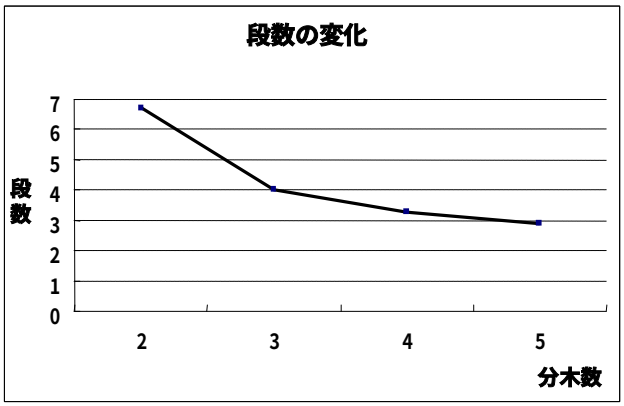


図 6. 各分岐数に対する段数の値

図 6 から計算上二分木から三分木にすることで, 約 7 段構成のクロック回路が, 約 4 段構成のクロック回路まで段数が 40%程度削減される可能性が見られる. しかし, 三分木より上では, 20%以下しか削減されない.

3　二分木と三分木の生成法

前章で n 分木における分木点数と段数でみられた傾向が実際のクロック回路でも成立するかどうかについて, 実際に木を構成することにより実験的に確認する.

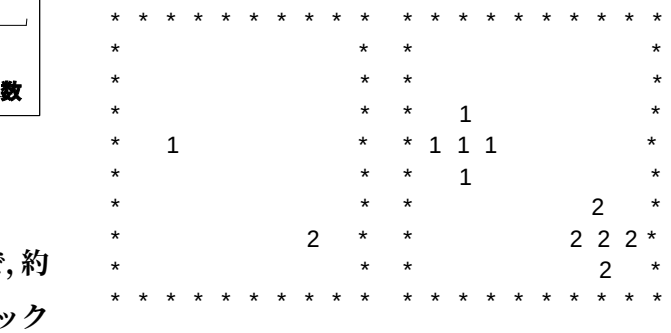
3.1 端子周囲の探索（ダイヤモンド型探索）

二分木の生成のために, 最隣接する端子ペアの探索を行う. まず, 各端子から周囲に各端子の番号をラベル付けし, これを繰り返す（ダイヤモンド型の端子領域を広げる）. 異なる端子のダイヤモンド領域が重複したものを最隣接ペアとして見つけ出すことができる.

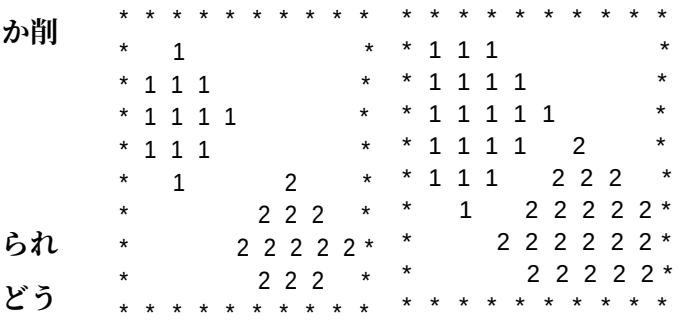
端子２つの場合のダイヤモンド型探索について, 途中の段階を含めて図 7 に 6 段階の図で示す.

まず, 端子番号 1, 2 を配線領域におく（図 7.1）. 次に, 周囲に１グリッドずつダイヤモンド型を広げる（図 7.2）. これを繰り返していく（図 7.3, 図 7.4）. 両端子のダイヤモンド領域が接した様子を図 7.5 に示す. 図 7.5 の段階で, １, ２が最寄のペアであることがわかるのでペアリングする. さらに, 両端子の中間点を上位の端子３として設定する.（図 7.6）

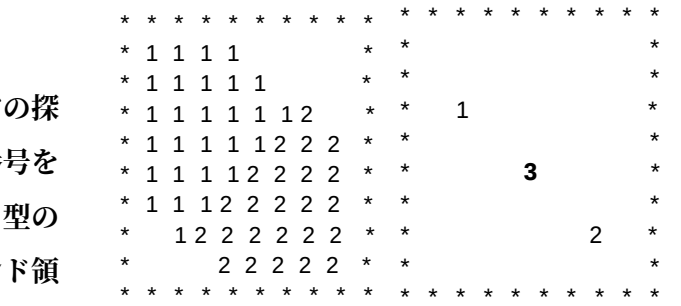
さらに上位の端子で繰り返すことで, 二分木が生成できる.



(1)　端子の配置　　(2)周囲への探索



(3)探索の継続１　　(4)探索の継続２



(5)2 端子の接触　　(6)上位端子の設定

図 7. ダイヤモンド探索

以上のアルゴリズムは以下のとおりである.

```
Binary Tree{
    if(map[x][y]!=BLANK){
        t2 = map[x][y];
            if(wt[t].p==OFF&&wt[t2].p==OFF){
                    maxwt++;
                    chuoyer(t,t2,&wt[maxwt].x,&wt[maxwt].y);
                    wt[maxwt].level=wt[t].level+1;
                    wt[maxwt].cn=2;
                    wt[t].p=maxwt; wt[t2].p=maxwt;
                     delmap(t); delmap(t2);
              }
    } else {
          map[x][y]=t;
    }
}
```

t,t2 がぶつかった端子番号である．t が map[x][y]の領域内で空白以外にぶつかれば，端子番号 t2 を記憶する．もし両方に親がいない場合であれば，2 分木を生成することができるので，最大端子数+1 を行い新たな親端子の番号を設定する. wt[ ]は端子である．chuoyer( )で t,t2 の(x,y)座標の中間値を計算し，この座標を親端子の座標とする．またこの座標は階層毎にペア生成が完了したら, 再度親の座標を割当てする．その時は，子の座標を全て足し，子の数で割ったものを親座標とする. これは二分木生成よりも三分木生成に有利である. 座標を設定すれば, 親のレベルを子のレベルより 1 上げ，子の数を 2 とする．t,t2 の親を設定し，二分木よりこれ以上ペア生成をする必要はないので，delmap( )でマップ上から t,t2 を削除する. またぶつからなかった場合そのままにし再度ダイヤモンド型領域を広げていく. ペア生成はその階層で端子数が 1 になるまで繰り返す.

なお端子数が奇数になる場合, 端子をすべてペアにすることができないが，その場合は，端子レベルを 1 あげることで，次の階層でペア生成する.

3.2 三分木生成法

二分木とほぼ同様に三分木の生成方法について説明する．三分木では，3 つの端子についてダイヤモンド探索し，ぶつかったものをグループとする．アルゴリズムは以下のとおりである.

```
Triplet Tree{
    if(map[x][y]!=BLANK){
        t2 = map[x][y];
          if(wt[t].p==OFF && wt[t2].p==OFF) {
                  maxwt++;
                  chuoyer(t,t2,&wt[maxwt].x,&wt[maxwt].y);
                   wt[maxwt].level=wt[t].level+1;
                  wt[maxwt].cn=2;
                  wt[t].p=maxwt; wt[t2].p=maxwt;
          } else if(wt[t2].p==OFF) {
                   wt[t2].p=wt[t].p; wt[wt[t].p].cn++;
          } else {
                   wt[t].p=wt[t2].p; wt[wt[t].p].cn++;
          }
          if(wt[wt[t].p].cn>2) {
                 for(i=1;i<=maxwt;i++) {
                    if(wt[i].p==wt[t].p) delmap(i);
                 }
           }
    } else {
         map[x][y]=t;
      }
}
```

t,t2 がぶつかった端子番号である．t が map[x][y]の領域内で空白以外にぶつかれば，端子番号 t2 を記憶する．もし両方に親がいない場合であれば，ペアを生成することができるので，最大端子数+1 を

行い新たな親端子の番号を設定する. wt[ ]は端子である．chuoyer( )で t,t2 の(x,y)座標の中間値を計算し，この座標を親端子の座標とする．座標を設定すれば，親のレベルを子のレベルより 1 上げ，子の数を 2 とする．t,t2 の親を設定する．三分木なので，また繰り返し，グループ生成を行う．次に t に親がいて t2 に親がいない場合 t2 の親を t の親と同じにする．そして t の子の数を増やす．また t2 に親がいて t に親がいない場合 t の親を t2 の親と同じにする．そして t の子の数を増やす．もし t の子の数が 2 より多くなれば，三分木生成は完了しているので，その端子を delmap( )より削除する．ぶつかりあわなければそのままにし再度ダイヤモンド型領域を広げていく．グループ生成はその階層で端子数が 1 になるまで繰り返す.

親の座標は階層毎にグループ生成が完了したら，再度親の座標を割当てする．その時は，子の座標を全て足し，子の数で割ったものを親座標とする.

なお三分木であるので，端子数が 3 のべき乗でない場合，端子をすべてグループにすることができないが，その場合は，端子レベルを 1 あげることで，次の階層でグループ生成する.

3.3 二分木と三分木の比較

概念的な図で，二分木と三分木の生成の過程を比較する．この図では端子数を 16 と設定する.

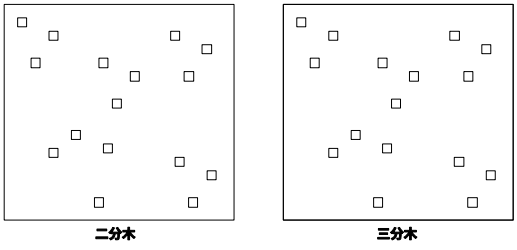


図 8．初期配置(level 0)

(1) level 0

最初に端子を配置する. 端子の位置は図 8 のように，比較するためお互い同じ配置とする.

(2) level 0

最寄の端子をグループ化する. 図 9 の三分木で端子グループが 1 の単独である端子が存在するが，これ以上グループ生成はできないのでそのままにしておく.

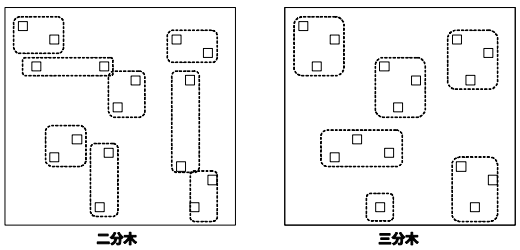


図 9．グループ生成

(3) level 0

グループ化した端子の中間点を親として上の階層に設定する. 図 9 の三分木でグループが 1 である単独の端子が存在したが，その端子は次の階層でグループ化させるので，そのまま階層を 1 あげる.

level0 で親端子は上の階層に設定してあるので，本来は存在していないが，中間点に配置したことを証明するため，図 10 より親端子の色を黒で設定してある. 今後黒で表記した端子は上の階層で設定した端子として表す.

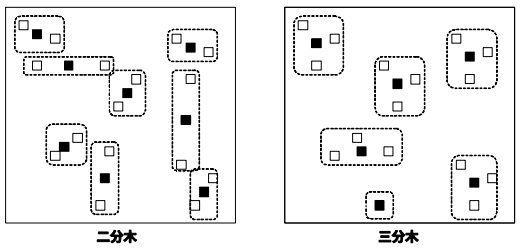


図 10．親端子設定

(4) level 0→level 1

グループ化と親端子の設定が全て終了すれば, 次の階層にあがる.（図 11）

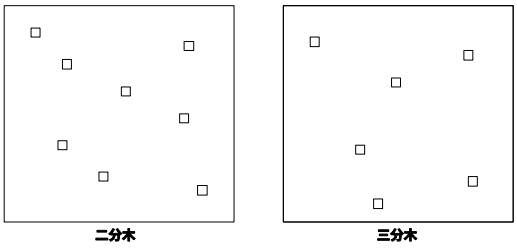


図 11. level 1

(5) level 1

グループ化し，図 12 のように親端子を設定する.

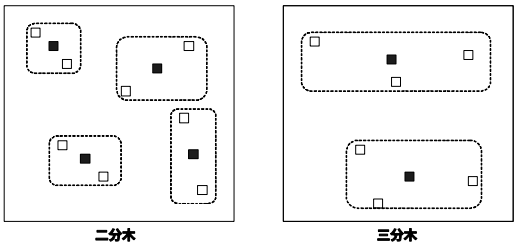


図 12. 親端子設定

(6) level 1→level 2

グループ化と親端子が全て終了すれば, 次の階層にあがる.（図 13）

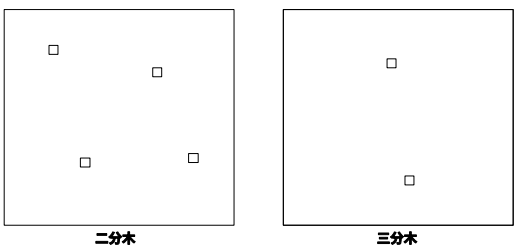


図 13. level 2

(7) level2

グループ化し親端子を設定する. 図 14 のように三分木グループの端子が２である場合, 二分木生成と同様に中間点に親を設定する.

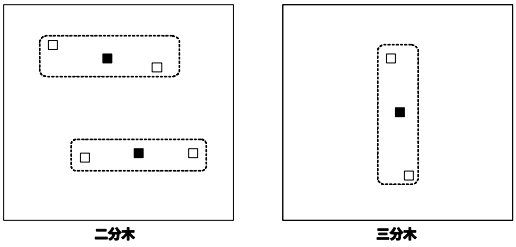


図 14. 親端子設定

(8) level2→level3

グループ化と親端子の設定が全て終了すれば, 次の階層にあがる. 図 15 の三分木は端子数が 1 となったので，グループ生成は終了となる.

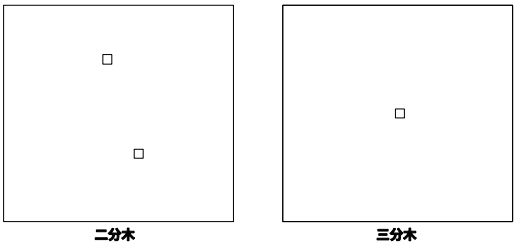


図 15. level3

(9) level3

グループ化し親端子を設定する.（図 16）

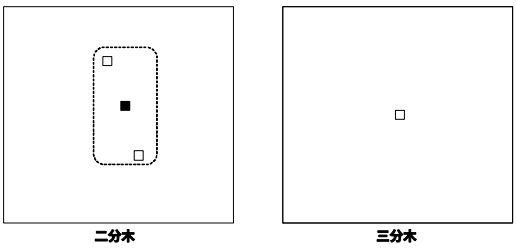


図 16. 親端子設定

(10)level3→level4

グループ化と親端子が全て終了し，図 17 の二分木

も端子数が 1 となったので, 両方のグループ生成は終了する.

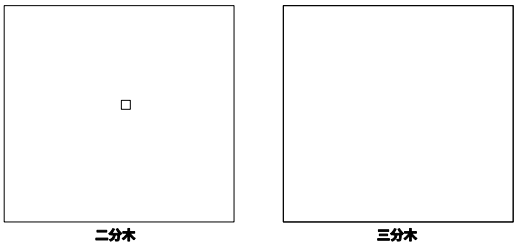


図 17. 端子生成終了

(11)概念図

グループ生成を行った結果, このような木構造となる.（図 18）

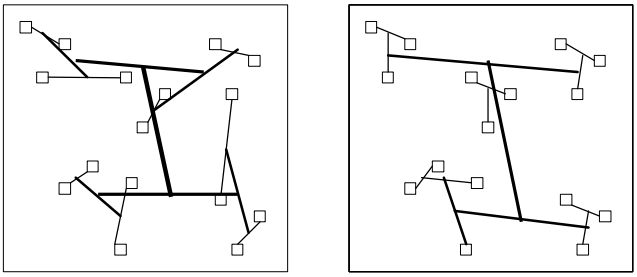
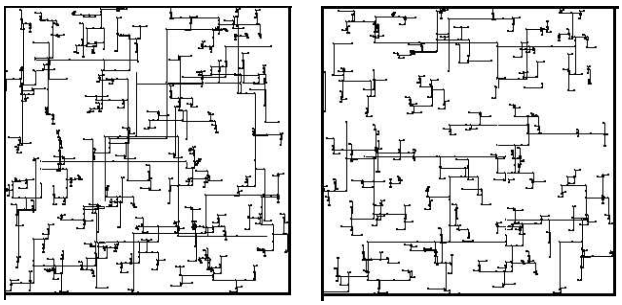


図 18. 概念図

図 19 はレイアウト領域 100×100, 端子数 100 で行い, 配線は豊永研究室の配線ツールを利用している. 実際の配線は以下のようになる.



(a)二分木　　(b) 三分木

図 19. クロック木の詳細配線(サイズ 100x100, 端子数 100)

実際比較すると, 三分木の方が段数も少なく, 詳細配線もすっきりして見える. より詳しい結果を実験で紹介する.

4 実験

最初にレイアウト領域と端子数を設定し, 配置に関しては乱数を用いてランダムに配置する. そこで配置による配線長の変化と二分木と三分木のクロック構造について総配線長と段数を比較する実験を行った. Cygwin の c 言語で実装した.

4.1 ランダム配置の端子生成

端子の配置は乱数を使用し, ランダムに行っている. 配置によって総配線長はどうなるか確認した. レイアウト領域 100×100 と 200×200, 端子数 100 と 200 で乱数を 1～50 を 10 毎に設定した. テストデータの概要を表 1 と表 2 に示す.

表 1. テストデータによる分木数毎の配線長の比較
(サイズ 100x100, 端子数 100)

	1	10	20	30	40	50	Av
Btree	1641	1546	1664	1648	1676	1512	1615
Ttree	1349	1395	1494	1368	1363	1437	1401

表 2. テストデータによる分木数毎の配線長の比較
(サイズ 200x200, 端子数 200)

	1	10	20	30	40	50	Av
Btree	4674	4722	4719	5042	5041	4807	4834
Ttree	4101	4001	4030	4063	4040	4030	4044

B-tree は二分木, T-tree は三分木, 1～50 は設定した乱数である. 結果は２つのクロック構造において, 端子の配置が変化しても総配線長はほぼ等しいことがわかる. よって配置は総配線長に影響しないことがわかった.

4.2 配線長・分木点数と段数

２つのクロック構造について, 総配線長および段数

（階層数)を調べる実験をおこなった．ベンチマーク用の端子データは，レイアウト領域 100×100～400×400 の 4 種類の規模に対して端子数 50～400 を 50 毎に配置した．総配線長は表３～6 となる．

表 3．総配線長比較　(サイズ 100x100)

#term	Btree	Ttree	length diff(%)
50	1083	982	9.3%
100	1641	1349	17.8%
150	2002	1743	12.9%
200	2368	1917	19.0%
250	2751	2262	17.8%
300	2999	2430	19.0%
350	3180	2718	14.5%
400	3512	2965	15.6%

表 4．総配線長比較　(サイズ 200x200）

#term	Btree	Ttree	length diff(%)
50	1916	1697	11.4%
100	3111	2882	7.4%
150	4048	3484	13.9%
200	4674	4101	12.3%
250	5432	4518	16.8%
300	5981	5008	16.3%
350	6724	5421	19.4%
400	7018	6046	13.9%

表 5．総配線長比較　(サイズ 300x300)

#term	Btree	Ttree	length diff(%)
50	3262	2938	9.9%
100	5341	4125	22.8%
150	5894	5305	10.0%
200	6927	6116	11.7%
250	8051	6655	17.3%
300	8896	7525	15.4%
350	9679	8023	17.1%
400	10762	8983	16.5%

表 6．総配線長比較　(サイズ 400x400）

#term	Btree	Ttree	length diff(%)
50	4168	3523	15.5%
100	6343	5506	13.2%
150	8486	6901	18.7%
200	9839	7812	20.6%
250	10574	8924	15.6%
300	11400	9652	15.3%
350	12893	10590	17.9%
400	14107	11285	20.0%

#term は端子数, length diff は総配線長の削減率となる．

結果は全てにおいて二分木より三分木の方が総配線長は削減されており, 7.4%～最大 20.6%の削減率が見られた. また総配線長はレイアウト領域と端子数によって左右されることがわかった.
次に分木点数の結果を表 7～11 に述べる．element diff は分木点数の削減率である．

実際の分木点数は二分木の場合全て同じ結果になった．三分木もほぼ同様の結果となっている．
二分木に比べ三分木の分木点数は削減できており, 30%以上の削減率が見られた.

表 7．分木点数比較　(理論値)

#term	Btree	Ttree	element diff(%)
50	49	24	51.0%
100	99	49	50.5%
150	149	74	50.3%
200	199	99	50.3%
250	249	124	50.2%
300	299	149	50.2%
350	349	174	50.1%
400	399	199	50.1%

表 8．分木点数比較　(サイズ 100x100)

#term	Btree	Ttree	element diff(%)
50	49	30	38.8%
100	99	61	38.4%
150	149	90	39.6%
200	199	129	35.2%
250	249	155	37.8%
300	299	183	38.8%
350	349	218	37.5%
400	399	248	37.8%

表 9．分木点数比較　(サイズ 200x200）

#term	Btree	Ttree	element diff(%)
50	49	31	36.7%
100	99	65	34.3%
150	149	92	38.3%
200	199	124	37.7%
250	249	156	37.3%
300	299	186	37.8%
350	349	210	39.8%
400	399	240	39.8%

表 10．分木点数比較　(サイズ 300x300)

#term	Btree	Ttree	element diff(%)
50	49	31	36.7%
100	99	65	34.3%
150	149	84	43.6%
200	199	129	35.2%
250	249	157	36.9%
300	299	190	36.5%
350	349	229	34.4%
400	399	254	36.3%

表 11．分木点数比較　(サイズ 400x400）

#term	Btree	Ttree	element diff(%)
50	49	34	30.6%
100	99	61	38.4%
150	149	100	32.9%
200	199	122	38.7%
250	249	162	34.9%
300	299	188	37.1%
350	349	216	38.1%
400	399	246	38.3%

表 3 の最初にある理論値は章 2.3 で計算した結果である。比較すると二分木は全て同じであり，三分木に関しては 20〜30%程度違いがあるが，本研究で使用した三分木は二分木や単独になる場合があるので多少理論値通りに 50%も削減することはできなかったが, それを踏まえて考えると理論値に近く，削減率は高いことがわかる．

分木点数は規模の濃淡によらず,端子数のみ関係があることがわかった．最後に段数の結果を表 12 に述べる．calculation は理論的に計算した値（理論値）である．実際の段数は全てにおいて同様の結果となった. しかし Ttree でレイアウト領域 300×300, 端子数 400 の時に段数が７，レイアウト領域 400×400，端子数 150 の時に段数が７となった.

本手法の三分木は二分木を生成して三分木を生成し, 三分木になれない端子は次の階層へあげる手法を用いているので, それが影響されていると考えられる.

段数の削減は二分木に比べ三分木は理論値が 2.3～3.5 段, 実際の段数は 2～3 段削減された. よって実際の段数は理論値とほぼ同等の結果が出ていると言えそうである.

表 12. 段数比較

#term	Btree	Ttree	Calculation	
			Btree	Ttree
50	6	4	5.7	3.4
100	7	5	6.7	4.0
150	8	5 (6)	7.3	4.4
200	8	6	7.7	4.6
250	8	6	8.0	4.8
300	9	6	8.3	5.0
350	9	6	8.3	5.0
400	9	6 (7)	8.7	5.2

5 まとめ

実験から従来の二分木構造のクロック回路に対して, 三分木構造では 2～3 段の段数削減が期待でき, 理論値とほぼ同等な削減ができることがわかった. そして分木点数も 30%以上削減でき, 理論値に近い削減率が見られることがわかった. また端子の位置のばらつきによらず, 10～20%の配線長削減が期待できる. よって二分木より三分木の方がクロック回路規模が縮小でき, 消費電力の削減に有益であると思われる.

今後の課題としては三分木以上におけるクロック木の実装方法の更なる検討が必要である.

謝辞

本論文の作成にあたって, ご指導頂きました豊永昌彦先生に深く敬意を表し, 深く感謝致します. また協力して頂いた同研究室の海老江 光さん, 吉田祐馬さんをはじめ, 同研究室の方々に感謝致します.

参考文献

[1]海老江光, 下川遥香, 豊永昌彦, “三分木に基づくクロック配線の一手法”, 電気関係学会四国支部連合大会, 2007, 1-5

[2]Ganesh Venkataraman, Zhuo Feng, Jiang Hu, Peng Li,“Combinatorial Algorithms for Fast Clock Mesh Optimization”ICCAD’06,November5-9,2006,San Jose, CA

[3] Moses Charikar, Jon Kleinberg, Ravi Kumar, Sridhar Rajagopalan, Amit Sahai, Andrew Tomkins, “Minimizing wirelength in zero and bounded skew clock trees” Proc. of 10th ACM-SIAM Symposium on Discrete Algorithms, pp.177-284, 1999.

[4]コンピューターの構成と設計 第２版
ジョン・L・ヘネシー ディビット・A・パターソン 日経出版 1999 年.