M.Tech. (Computer Science) Dissertation Series

A general recursive staircase bipartition scheme for VLSI floorplan layout with simultaneous minimization of net crossovers

A dissertation submitted in partial fulfillment of the requirements for the M.Tech. (Computer Science) degree of the Indian Statistical Institute

By

Amlan Bag

Under the supervision of

Prof. Bhargab B. Bhattacharya

INDIAN STATISTICAL INSTITUTE

203, B. T. Road

Kolkata - 700108

Synopsis

A general recursive staircase bipartition scheme for VLSI floorplan layout with simultaneous minimization of net crossovers.

Submitted by

Amlan Bag

Motivation:

In DSM (deep submicron) VLSI, interconnect delay plays a major role in determining system performance, reliability and cost. To ensure timing closure of designs, impacts of interconnect should be incorporated as early as possible in the design flow. There are several approaches to mitigate the interconnect delays. Repeater insertion is widely accepted approach to minimize delay.

Our work is primarily a recursive bi-partitioning of a floorplan layout with staircase channels which is aware of repeater overlap and congestion. This scheme doesn't directly deals with repeater placement but it reveal a global interconnect planning which will facilitate repeater placement in routing stage. The basic bi-partition algorithm is primarily based on *stair-case bipartition* scheme proposed by Dasgupta et.all. Some modification is incorporated to meet our requirement. This is a top-down approach. The algorithm recursively divides the floorplan into two equal halves with a monotone staircase path or staircase channel. At each level of the hierarchical channel, using a simple channel scan based heuristic, cross over is minimized. Reason is to reduce the overlap of repeater form different nets that are to be inserted in the channel. It is observed that pins on both side of channel which is in the deepest level of partitioning (it is those channel which separates only two adjacent blocks) are to be connected individually. And going higher up the channel order it is sufficient to connect any two pins of same net form either side of the channel. This considerably reduces routing overhead. It is also worth mentioning, this is a general bipartition scheme as it works equally well for both slicing and *non-slicing floorplan.*

Problem formulation:

Definition: A monotone staircase path $P= \{l_1, l_2, l_3, l_4 \ldots l_n\}$ is a set of line segments such that (i) two line segment l_i, l_{i+1} share a common end and they are either horizontal or vertical to each other. (ii) let $a_i=(x_i,y_i)$ and $a_{i+1}=(x_{i+1},y_{i+1})$ be two point belonging to line segment l_i, l_{i+1} respectively , then $\forall$ i either (1). $x_i \leq x_{i+1}$ and $y_i \leq y_{i+1}$

or (2) $x_i \leq x_{i+1}$ and $y_i \geq y_{i+1}$

Problem formulation : Given a rectangular floorplan $F=\{R_1,R_2,R_3 \ldots .R_m\}$ find a set of hierarchical monotone staircase path $P_F= \{(p_1),(p_{11},p_{12}),(p_{111},p_{112},p_{121},p_{122}) \ldots .\}$ such that

i) p_1 divides F into two half say $F_{P1}1$ and $F_{P1}2$ with either equal number of blocks or one extra in any one of the half (equal if even number of blocks in the floorplan) and the next level of path p_{11},p_{12} divides $F_{P1}1$ and $F_{P1}2$ further maintaining the equality condition. This partitioning continues until the given floorplan is disintegrated to individual rectangles.

ii) The cross-over of nets in each hierarchical path is minimized.

Overview of the work:

In total there are four steps: i) parsing of the input file. ii) *Floorplan graph generation* from the parsed data. iii) Generation of *slicing tree* with *staircase channels*– this main part of the procedure. iv)Minimization of crossing of nets in each channel. Three procedures (ii, iii & iv) are used simultaneously and recursively to divide the floorplan individual rectangular blocks and a complete staircases slicing tree is generated. The leaf nodes of this tree are the rectangle blocks of the floorplan.

A brief explanation of the individual steps:

i) Floorplan graph generation: finds out the *adjacency* (both *horizontal* and *vertical*) between rectangular blocks of the floorplan with some prefixed ordering of the rectangular blocks.

ii) Slicing tree generation: which consist of several sub-procedures –sequentially they are ii.a) block labeling – which is a modified version labeling algorithm proposed by Dasgupta et. all ii.b) euipartioning the floorplan into two halves based on labeling. ii.c) finding the staircase channel. ii.d) extracting those nets which falls in to the staircase channel from the connection netlist to apply next procedure.

iii) Crossing minimization of nets: a channel scan based heuristic that tries to minimize the crossover of different nets.

Algorithmically overall process is:

Input: i) floorplan i.e. individual rectangle blocks with coordinates of left-bottom corner point and heights and widths.

ii) Connection net list.

Output: A staircase slicing tree with minimized crossovers in each individual staircase channel.

Procedure:

1. Parse the input file and store the data into a floorplan data-structure.
2. While(complete slicing tree is not achieved){
   3. Generate slicing tree {
      // This consist of several steps
      i) generate floorplan graph with respect to a predefined direction
      ii) label the floorplan graph.
      iii) find the staircase channel that divides the floorplan
      iv) extract the net information for the channel .
      v) do the crossover minimization.
      vi) keep the two half of the divided part in separated floorplan data structure and apply step2 to both half.
4. }End while.

Result :

The algorithm has been tested on some MCNC floorplan benchmarks and some randomly generated nonslicing floorplan. The tabulated result shows the circuits and the number of channel that it finds while portioning and the overall computation time.

Name of the floorplan (MCNC/ Randomly generated)	Number of staircase channels	Number of CPU cycle
Xerox	9 # of Level 0 channel -1 # of Level 1 channel -2 # of Level 2 channel -4 # of Level 4 channel -2	281
hp	Same as Xerox	357
n10	do	320
Randomly generated nonslicing floorplan nsl1	5 # of Level 0 channel -1 # of Level 1 channel -2 # of Level 2 channel -1	156
Randomly generated nonslicing floorplan nsl2	8 # of Level 0 channel -1 # of Level 1 channel -2 # of Level 2 channel -4 # of Level 4 channel -1	250

Conclusion:

Here we presented an approach for recursive partitioning of floorplan and simultaneous minimization of crossover of nets. A simple heuristics for minimizing the net crossover is proposed here. The proposed approach has certain advantages (i) a *safe routing order* is always achieved for both nonslicing and slicing floorplan. (ii) Staircase path is targeted to partition a floorplan in almost equal halves with respect to the number of blocks in each level of recursion; the depth of hierarchy tends to be smallest. (iii) Since at level of hierarchical channel minimum number of pin is connected that simplifies the routing. Though overall time complexity is on the higher side we are investigating ways of reducing running time.

Key Words: **VLSI, Bipartition, Staircase Channel, Buffer insertion, Net cross over**

Table of Contents

Chapter 1

Introduction:

Present day VLSI circuit dimensions are reduced to a deep submicron region. Continued technology scaling has minimized intra circuit delays comprehensively. Whereas due to inherent limitation of connecting wires, interconnect delays are considerably high. These interconnect delays plays a major role in determining the system performance, reliability, and cost. To ensure timing closure of designs, impacts of interconnect should be incorporated as early as possible in the design flow. Several approaches such as – topology construction, repeater insertion, wire sizing, device sizing has been studied. Repeater insertion is widely accepted approach to optimize signal delay.

The lumped capacitance and resistance of the wires is the contributor to this delay. For an interconnect of length l with resistance r Ω per unit length and capacitance c μF per unit length, by Elmore delay calculation the delay is approximately $r.c.l^2$ [1]. By appropriate insertion of repeaters, which is a combination of inverting and non-inverting buffers, delay becomes linear with the length [2]-[4]. A large scale integration design must have certain objectives when it comes to repeater insertion. These design objectives are (i) to minimize interconnect delay and (ii) limit transition times while limiting impact on (a) area and (d) power. Previous work like [5] has considered random placement. However repeater consumes silicon resources and there are certain circuit blocks such as cache where repeater is not allowed. To mitigate this problem insertion of repeater in channels is also considered. However the greedy clustering of repeaters may end up in increased buffer congestion. As suggested in [6] that interconnect planning for global routing should be done in the

floorplanning stage. Floorplanning is the stage of design where blocks are positioned in the layout surface; in such a fashion that no two blocks are overlapping and enough space is left on the layout surface to complete the interconnection [8,sherwani- "*Algorithms for VLSI physical design automation*", 3rd edition]. Connection netlist is also available in this layer.

Our work is primarily a recursive bi-partitioning of a floorplan layout with staircase channels which is aware of repeater overlap and congestion. This scheme doesn't directly deals with repeater placement but it reveal a global interconnect planning which will facilitate repeater placement in routing stage. The basic bi-partition algorithm is primarily based on *stair-case bipartition* scheme proposed by Dasgupta et. all[7]. Some modification is incorporated to meet our requirement. This is a top-down approach. The algorithm recursively divides the floorplan into two equal halves with a monotone staircase path or staircase channel. At each level of the hierarchical channel, using a simple channel scan based heuristic, cross over is minimized. Reason is to reduce the overlap of repeater form different nets that are to be inserted in the channel. It is observed that pins on both side of channel which is in the deepest level of partitioning (it is those channel which separates only two adjacent blocks) are to be connected individually. And going higher up the channel order it is sufficient to connect any two pins of same net form either side of the channel. This considerably reduces routing overhead. It is also worth mentioning, this is a general bipartition scheme as it works equally well for both slicing and *non-slicing floorplan.*

Chapter 2

Problem definition:

Definition: A monotone staircase path P= $\{l_1, l_2, l_3, l_4 \ldots l_n\}$ is a set of line segments such that (i) two line segment l_i, l_{i+1} share a common end and they are either horizontal or vertical to each other. (ii) let $a_i=(x_i,y_i)$ and $a_{i+1}=(x_{i+1},y_{i+1})$ be two point belonging to line segment l_i, l_{i+1} respectively , then $\forall$ i either (1). $x_i \leq x_{i+1}$ and $y_i \leq y_{i+1}$

or (2) $x_i \leq x_{i+1}$ and $y_i \geq y_{i+1}$

1.1(a) 1.2(b)

Fig 1.1: (a) is increasing monotone staircase path.

(b) is decreasing monotone staircase path.

Problem formulation : Given a rectangular floorplan F=$\{R_1,R_2,R_3 \ldots..R_m\}$ find a set of hierarchical monotone staircase path P_F= $\{(p_1),(p_{11},p_{12}),(p_{111},p_{112},p_{121},p_{122})\ldots.\}$ such that

i) p_1 divides F into two half say $F_{P1}1$ and $F_{P1}2$ with either equal number of blocks or one extra in any one of the half (equal if even number of blocks in the floorplan) and the next level of path p_{11},p_{12} divides $F_{P1}1$ and $F_{P1}2$ further maintaining the equality condition. This partitioning continues until the given floorplan is disintegrated to individual rectangles.

ii) The cross-over of nets in each hierarchical path is minimized.

Chapter 3

Overview of the flow of work:

In this section an outline of the overall flow of work is given. Details of each steps and algorithms are explained in the next section. In total there are four steps: i) parsing of the input file. ii) *Floorplan graph generation* from the parsed data. iii) Generation of *slicing tree* with *staircase channels*– this main part of the procedure. iv)Minimization of crossing of nets in each channel. Three procedures (ii, iii & iv) are used simultaneously and recursively to divide the floorplan individual rectangular blocks and a complete staircases slicing tree is generated. The leaf nodes of this tree are the rectangle blocks of the floorplan.

A brief explanation of the individual steps:

i) Floorplan graph generation: finds out the *adjacency* (both *horizontal* and *vertical*) between rectangular blocks of the floorplan with some prefixed ordering of the rectangular blocks.

ii) Slicing tree generation: which consist of several sub-procedures –sequentially they are ii.a) block labeling – which is a modified version labeling algorithm proposed by Dasgupta et. all [7]. ii.b) euipartioning the floorplan into two halves based on labeling. ii.c) finding the staircase channel. ii.d) extracting those nets which falls in to the staircase channel from the connection netlist to apply next procedure.

iii) Crossing minimization of nets: a channel scan based heuristic that tries to minimize the crossover of different nets.

Algorithmically overall process is:

Input: i) floorplan i.e. individual rectangle blocks with coordinates of left-bottom corner point and heights and widths.

ii) Connection net list.

Output: A staircase slicing tree with minimized crossovers in each individual staircase channel.

Procedure:

1. Parse the input file and store the data into a floorplan data-structure.
2. While(complete slicing tree is not achieved){
    3. Generate slicing tree {
        // This consist of several steps
        i) generate floorplan graph with respect to a predefined direction
        ii) label the floorplan graph.
        iii) find the staircase channel that divides the floorplan
        iv) extract the net information for the channel .
        v) do the crossover minimization.
        vi) keep the two half of the divided part in separated floorplan data structure and apply step2 to both half.
4. }End while.

Chapter 4

Detailed description of the work:

4.1 Input file parsing:

The first step is parsing of the input file and storing the retrieved data into a suitable data-structure which we call floorplan data-structure. The input file is written in a simplified YAL format. The grammar of the file-format is given below:

< blocks>

< nets>

module

<module name> <x1> <y1> <height> <width>

pin

<pin_id> <side> <x1><y1> <net>

:

:

:

p_end

end

<blocks> := integer , specifying the number of individual blocks present in the file .

<nets> := integer , specifying the number of different nets.

module :

Definition: a module is definition of a circuit is being laid out or a constituent (or primitive cell). The definition of each module starts with the key word “module” and ends with “end”. A template of a module is as follows:

<module name> := is character string specifying id of the block.

<x1> <y1> := real numbers , is the coordinates of the left-most corner point.

<width> <height>:= real numbers, bears the usual meaning .

pin and p_end : marks the start and end of the pins or input/output terminals.

<pin_id>:= character string, specifying the id.

<side>:= single character. Specifies whether the pin is located on the left-side or right-side or on top or in the bottom of a rectangular block.

<side>:='0'|'2'|'4'|'6'.

'0'=bottom.

'2' and '4'-denotes left and right side respectively.

'6'=top.

<x1><y1>:= real numbers. Denote location.

<net>:=integer. Define connectivity of the pin.

Procedure: input_file_reader

Input: floorplan and netlist information given in YAL format.

Output: storage of data in floorplan data-structure.

begin:

1. *number_of_blocks ←<blocks>;*
2. *nets←<nets>;*
3. *string s←next input from the input-file.*
4. *if (s== "module"){*
   char c←'m';}
5. *while(!end of file){*
   switch(c){
   case 'm':
   1. *scan the input from the file and store the coordinates and height and width in local variables.*
   2. *Call procedure create-block (*floorplan, x1, y1, height, width) to create individual blocks.*
   3. *Scan next input and store it in s.*

if(s=="pin"){
c←p;}

break;

case 'p':

1. *Scan the inputs and store it in the netlist data-structure.*
2. *Scan next.*
   If(s=="p_end") then c←e;
   else c←p;

break;

case 'e':

if there is no more module we have reached the end of the file other-

wise go to the first state i.e. 'm'.

break;

}end of switch.

} end of while.

6. End of procedure input_file_reader.

This procedure is simulating deterministic finite automata, where cases denote its states.

4.2 The floorplan data-structure:

Earlier we have mentioned a data-structure named floorplan. This is a simple array of structure (*structures* and *array* bears the meaning as described in 'C') where each array element is a structure named block. A detail of the structure of the block is given below. (Using 'C' language notation):

```
Structure block {
char id[]; //name of the module;
point  left_bottom; // point is also a structure which consist of two floating point number to store the coordinates.
point right_bottom;
point right_top;
```

```
point left_top;
*block left; // this is a pointer to the next block. These pointers will be used to capture the adjacency.
*block right; // left is for vertical and right is for horizontal adjacency.
int label // for block numbering scheme.
int is_lebeled.
int is_right, // if adjacent block is found then depending on priority it can 1,2,3 other wise 0. This is needed
                    // for block numbering .
int is_left;
int netlist_indicator. // this denotes the location in the netlist where the pin information  of this block is kept.
}
```

Using array of structure gives a constant time access to each individual rectangle blocks of the floorplan.

4.3 Floorplan graph generation:

The second step is to generate a floorplan graph form the data that has been parsed earlier. As mentioned before, rectangles are used to represent circuit blocks in a layout design. Note that no two rectangles in plane are allowed to overlap. Rectangles may share edges i.e. two rectangle may be adjacent to each other. Given a set of rectangles R= $\{R_1, R_2, R_3 \ldots R_m\}$ corresponding to a layout in a plane, a *floor plan graph* is graph G= (V, E), where

V= $\{v_i|$ v_i represents the rectangle block R_i $\}$and

E= $\{(v_i, v_j)|$ if R_i and R_j is either vertically or horizontally adjacent$\}$

The construction of *floorplan graph* is the basis as all the subsequent procedure will be applied on the *floorplan graph.*

Algorithm for generating floorplan graph is given next.

Procedure: gen_fp_graph / for generating floorplan graph */*

Input: i) a floorplan array // explained earlier.

ii) A predefined direction – dir, according to which the adjacency will be traced.

iii) number of blocks in a floorplan.

Output : a floorplan graph each node of the graph is the individual blocks.

begin

1. switch(dir){

case '0':

1. Arrange the blocks according x-coordinates of their left- bottom corner points.

2. Find vertical adjacency of blocks, save the adjacency. (Procedure search_block1)

3. Arrange the blocks according y-coordinates of their left-bottom corner points.

4. find horizontal adjacency of the blocks, save the adjacency. (Procedure search_block2)

case '2':

1. Arrange the blocks according y-coordinates of their left-top corner points.

2. Find vertical adjacency of blocks, save the adjacency. (Procedure search_block2)

3. Arrange the blocks according x-coordinates of their left-top corner points.

4. Find horizontal adjacency of the blocks, save the adjacency. (Procedure search_block1)

case '4':

1. Arrange the blocks according x-coordinates of their right- bottom corner points.

2. Find vertical adjacency of blocks, save the adjacency. (Procedure search_block1)

3. Arrange the blocks according y-coordinates of their left-bottom corner points.

4. Find horizontal adjacency of the blocks, save the adjacency. (Procedure search_block2)

case '6':

1. Arrange the blocks according y-coordinates of their right-top corner points.

2. Find vertical adjacency of blocks, save the adjacency. (Procedure search_block2)

3. Arrange the blocks according x-coordinates of their right-top corner points.

4. Find horizontal adjacency of the blocks, save the adjacency. (Procedure search_block2)

} end switch.

2. end

The cases denote different direction .There is a reason to find out adjacency with respect to some certain directions. The overall process is recursive bi-partitioning and the cut is being a monotone staircase channel which runs from one corner point to the opposite corner point of a floorplan. Once a channel is found the floorplan is subdivided in two and the next levels of channels that would further divide these floorplans should be orthogonal to this channel. For example if cut a is directed left-bottom corner point to right-top corner point then the next level cuts (number of cuts is 2) will be directed form left-top to right-bottom corner point and vice versa. And to find these channels which are coming in from different direction we need to trace the adjacency with respect to different directions. Case '0'-denotes *left-bottom to right- top*, case '2'- denotes *left-top to right-bottom*, case '4'-denotes *right-bottom to left-top and case* '6'-denotes *right-top to left-bottom*. Throughout the report this convention has been maintained. Interestingly enough the overall orientation of the cuts can be expressed with help of a binary tree.

The adjacency information is kept in the floorplan data-structure. So the nodes of a floorplan graph are the rectangular blocks itself.

4.4 Adjacency of rectangular blocks:

In the preceding algorithm two procedures named *search_block1* and *search_block2* have been used interchangeably depending on the direction to find vertical and horizontal adjacency.

Two blocks can be vertically adjacent if it follows any one of the cases depicted in figure 4.2.(a) The condition for which blocks in case '0' and case '6'are vertically adjacent is true for horizontal adjacency in case '2' and case '4'. Similarly the conditions for horizontal adjacency in '0', '6' are the conditions for vertical adjacency in '2'and '4' (the three possible cases of horizontal adjacency are depicted in the figure 4.2(b)). The procedure *search_block1* traces vertical adjacency for case '0'and '6'while the same procedure traces the horizontal adjacency for case '2'& '4'. The second procedure, search_*block2,* dose exactly opposite of the first one. Priority has been imposed to the horizontal adjacencies- case I is of the highest priority next is case II and lowest III and the priority information is captured in is_*right* which is an element of the structure block. This is to maintain the monotonic property of the cut while numbering the blocks.

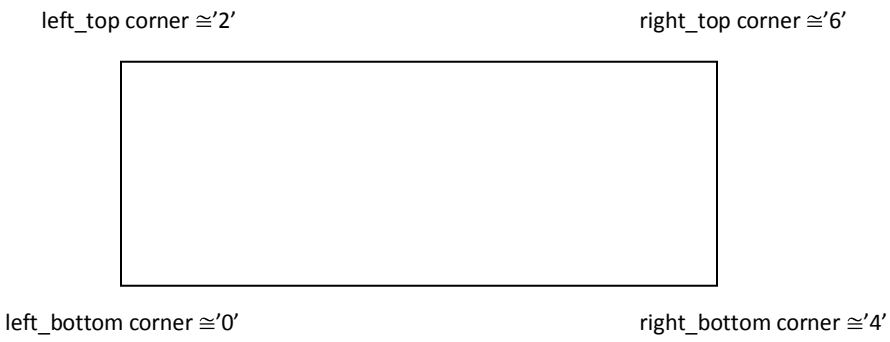
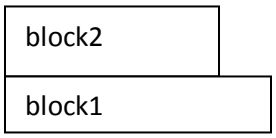


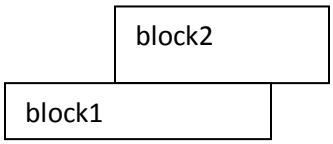
Fig. 4.1: a rectangular block

Vertical adjacency

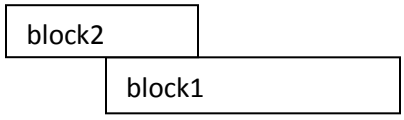
Case I



Case II.



Case III.



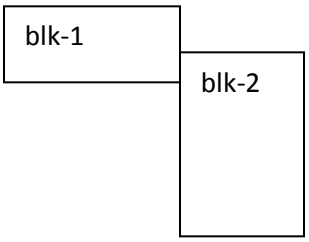
For '0' block2 is vertical to block1
For '6' block1 is vertical to block2
For '4' block2 is horizontal to block1
For '2' block1 is horizontal to block2

Fig : 4.2(a)

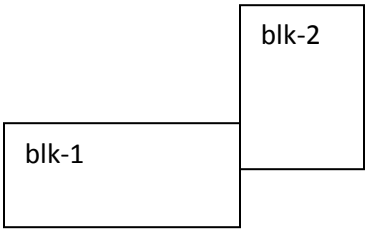
Horizontal adjacency

Case I

Case II



Case III



For '0' block2 is horizontal to block1
For '6' block1 is horizontal to block2
For '4' block1 is vertical to block2
For '0' block2 to is vertical to block1

Fig: 4.2(b)

Fig. 4.2 (a), (b): depicts the possible cases of adjacency

4.5 Block labeling:

Procedure : blk_label

Input : floorplan graph and an integer i.

Output : all nodes of the graph is labeled;

begin

*1. if(root!=NULL){ /*root denotes a node in the floorplan graph*/*

blk_label(root→left,0)

if((i= = 0) &(root is not labeled)){

root.→label←label;

mark the node as labeled

label++;}

if(root→right is not lowest in priority){

blk_label(root→right,0);}

}

If there are n number of blocks is present in the floorplan all the blocks will be numbered form 1to n. Blocks whose label is $\leq \Gamma(n/2)$, where n is the number of blocks in the floorplan , are in one half and the rest is in another half. Before applying the same procedures once again to these newly formed floorplans, the dividing staircase channel is traced out and simultaneously pins belonging to different nets which reside in either side of the channel are extracted so that net-cross-over can be minimized.

An example of floorplan lay out with its corresponding floorplan graph if the staircase channel moves form left-bottom corner point to right-top corner. Labeling of the floorplan and the channel is also traced out.

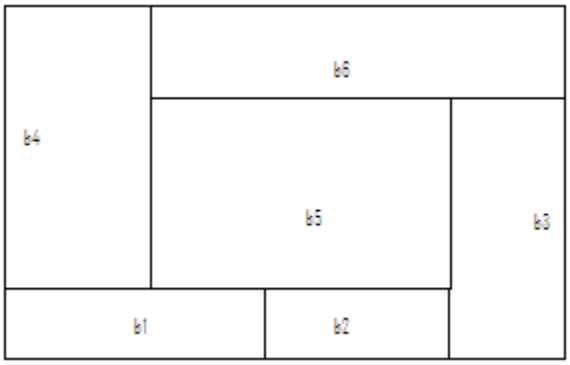


Fig. 4.3 (a) rectangular floorplan.

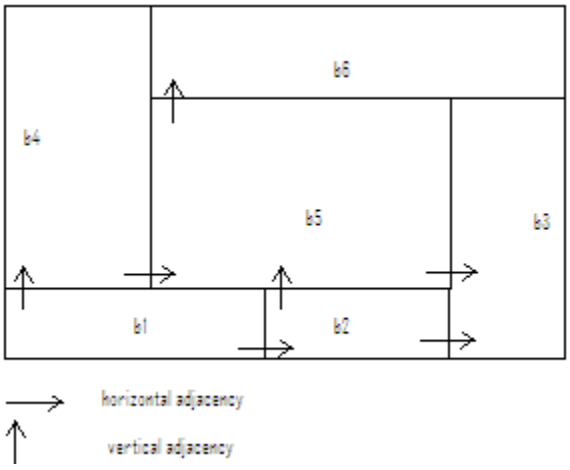


Fig. 4.3(b) Floorplan graph.

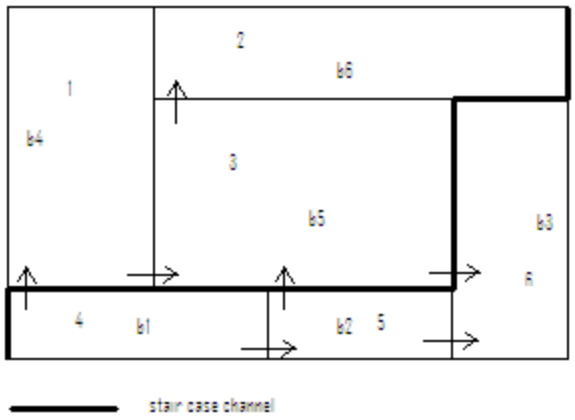


Fig. 4.3(c). Floorplan graph with labels and the staircase channel.

4.6 Staircase channel:

Definition of a staircase channel is given in the chapter 2 where we formally defined the problem. The algorithm is presented here. Basic principle is that it starts form the root of the floorplan graph and if the label of a node is greater than $\Gamma\ (^{n}/_{2})$ (n is the number of blocks the floor plan) then the channel moves vertically keeping the rectangular block in the right side of the channel otherwise it moves horizontally keeping the block in left side.

Detail procedure to find a staircase channel which runs from left-bottom corner point to right-top corner point, is given below.

Procedure: extract_net_sd0

Input : i) a labeled floorplan graph, to be specific root of the floorplan graph=fpg .

ii) number of nodes in floorplan graph, which is equal to the number of blocks present in the floorplan=n.

Output: a staircase channel is traced out and the pins on either side of the channel is extracted for net- crossover minimization.

begin

```
1. root←fpg;
2. previous←null
3. point  p ←fpg->left_bottom point;
4. if(root->label ≤ Γ(n/2))
        then movement ←h; // h stands for horizontal movement and v is for vertical movement
   else    movement ←v;
5. while (finished!=1){
        switch(movement){
                case v :
                    1. pins those are in between point p and root->left_top are in lower side of the
                       channel.
                     while(end!=1){
                          if(previous!=null){
                                if(previous->right_bottom  point ≥ root->left_top point ){
                                   pins on the right side of the pervious that are above point
                                   p are on the upper side of the channel.
                                  end=1;  }
                                else{
                                   pins on the right side of the pervious that are above point
                                   p are on the upper side of the channel.
                                  if(previous->left!=null) then
                                   previous←previous->left;
                                  else end=1;  }
                           } else end=1;
                     } end of while.
            if(root->left!=null)
```

```
{   if(root->left->label> Γ(n/2))
        {   movement←v;
            root←root->left;
            p←root->left_bot;
            end=0;   }
      else {   movement←h;
            previous←root;
            root←root->left;
            p←previous->left_top;
          end=0;     }
    }
   else {
        pins located on the top side of rectangular blocks are on the lower
        side of the channel
       finish=1;}
  break;
case h:
      pins, located on the bottom side  of root, which are in between point p and
      root->right_bottom point are on the upper side of the channel.
      while(end!=1){
        if(previous!=null){
            if(previous->right_top point ≥ root->right_bottom point){
                pins located on top side of previous and are in between root->left_bottom
                and root->right_bottom point is in the lower side of the channel.
                end←1; }
            else {
```

```
                pins located on top side of previous and are in between root->left_bottom
                and root->right_bottom point is in the lower side of the channel.
                    if(previous->right!=null){
                        previous← previous->right;
                        p← previous->left_top; }
                    else end←1; }
            } else end=1;
            if(root->right!=null){
                if(root->right->label > Γ(n/2)){
                    movement ←v;
                    previous ←root;
                    root ←root->right;
                    p ←previous->right_bottom;
                    end ←0;}
                else{   movement ←h;
                    root ←root->right;
                    p ←previous->right_bottom;
                    end ←0;}
            }
            else {
                pins on the right side of the root are in the upper side of the channel.
                finish=1;}
            break;
        }
    }
```

Next step is connecting the nets that falls in the channel in such a way that cross over between different nets is minimized if not zero.

4.7 Minimization of cross-over between nets:

This is a heuristics. The distribution of nets over a channel is not previously known so optimum result may not always be achieved. Basic principle is based on scanning of channel. It consist of two pass in one pass it scans form left to right and in the next pass it scans in opposite direction and whichever pass yields minimum result it finalizes those connections.

Procedure: cross_detect_min

Input: pin information for either side of the channel.

Output: minimized net cross-over connectivity of pins.

begin

keep a indicator for each net . if pin belonging to same net is present on both side of the channel then only connection is possible else those pin will remain floating.

*pass 1 /*direction is left to right*/*

1. for each not connected net on the upper side of the channel

start finding corresponding pin belonging to same in the opposite side in such way that

i) search always starts form the right-most connected pin.

ii) if found then connect and mark as connected and update the right-most visited pin or else continue searching till it hits the end.

iii) after hitting the end if still required pin is not found then move backwards form the right-most pin towards the left end till it finds the pin and every time it crosses a connected pin updated the number of cross over.

2. if net is already connected then ignore it.

3. *store the connectivity and number of cross-over.*

4. *end of pass1*

Pass 2 / direction is from right to left*/*

5. *exactly same procedure but in right to left direction.*

6. *compare the number of cross-over in each phase and select the minimum most and finalize the connection.*

7. *end .*

An example: Consider the channel and the distribution of pin in the figure

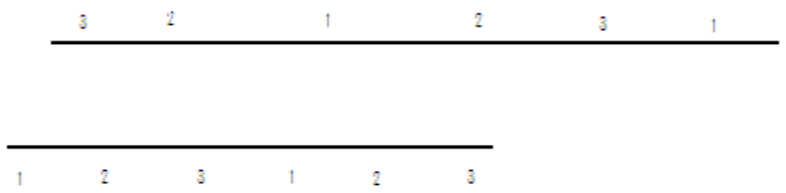


Fig 4.4 (a): channel with nets on both sides.

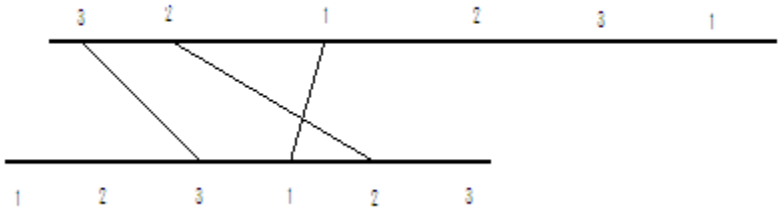


Fig 4.4 (b): after first pass of the algorithm with crossover =1.

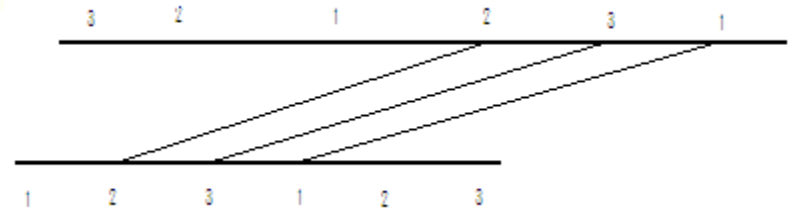


Fig 4.4(c): after second pass with 0 crossovers.

4.8 Slicing tree generation:

All the previous mentioned procedures (section 4.3, 4.5, 4.6, 4.7) are embedded in the procedure named gen_stair_tree and called recursively. Each node of the tree contains information regarding channel and their orientation, next level of nodes, position of the current node with respect to its parent node (i.e. left child or right channel). Several other information like crossover minimized interconnections that are achieved, are needed to be stored in each level of nodes that gives a hierarchical view of the overall routing.

Procedure: gen_stair_tree

Input: 1. Floorplan

2. Number of blocks in the floorplan

3. Direction as in the gen_fpgraph.

4. An integer specifying the location of the tree node (0-denotes the root of the slicing tree,

1- node is the right child and -1denotes left child of its parent).

Output : a node of slicing tree.

begin

switch (dir){

case'0':

if(n≥2){

1. call gen_fpgraph(floorplan, number of block, '0');

2. call blk_label (floorplan graph, number of blocks=n);

3. call extract_net_sd0(floorplan ,number of blocks=n);

4. call cross_detect_min;

5. bi-partitioning of floorplan: $\forall R_i \in F$ *if block label* $> \Gamma(n/2)$ *then*

R_i *is in right _half and right = right+1;else* R_i *is in left_half and left=left+1;*

6. create a node of the staircase tree. Store the necessary information in the tree node.

7. call gen_stair_tree(left_half, left,'2', -1);

8. call gen_stair_tree(right_half, right,'4', 1);

}

break;

case'2':

if(n≥2){

1. call gen_fpgraph(floorplan, number of block, '2');

2. call blk_label (floorplan graph, number of blocks=n);

3. call extract_net_sd2(floorplan ,number of blocks=n);

4. call cross_detect_min;

5. bi-partitioning of floorplan: $\forall R_i \in F$ *if block label* $> \Gamma(n/2)$ *then*

R_i *is in right _half and right = right+1;else* R_i *is in left_half and left=left+1;*

6. create a node of the staircase tree. Store the necessary information in the tree node.

7. call gen_stair_tree(left_half, left,'6', -1);

8. call gen_stair_tree(right_half, right,'0', 1);

}

break;

case'4':

if(n≥2){

1. call gen_fpgraph(floorplan, number of block, '4');

2. call blk_label (floorplan graph, number of blocks=n);

3. call extract_net_sd4(floorplan ,number of blocks=n);

4. call cross_detect_min;

5. bi-partitioning of floorplan: $\forall R_i \in F$ *if block label* $> \Gamma(n/2)$ *then*

R_i *is in right _half and right = right+1;else* R_i *is in left_half and left=left+1;*

6. create a node of the staircase tree. Store the necessary information in the tree node.

7. call gen_stair_tree(left_half, left,'0', -1);

8. call gen_stair_tree(right_half, right,'6', 1);

}

break;

case'6':

if(n≥2){

1. call gen_fpgraph(floorplan, number of block, '6');

2. call blk_label (floorplan graph, number of blocks=n);

3. call extract_net_sd6(floorplan ,number of blocks=n);

4. call cross_detect_min;

5. bi-partitioning of floorplan: $\forall R_i \in F$ *if block label* $> \Gamma(n/2)$ *then*

R_i *is in right _half and right = right+1;else* R_i *is in left_half and left=left+1;*

6. create a node of the staircase tree. Store the necessary information in the tree node.

7. call gen_stair_tree(left_half, left,'4', -1);

8. call gen_stair_tree(right_half, right,'2', 1);

}

break;

} finish.

Chapter 5

Results:

- The procedures are implemented using C on windows platform. (M/C specification 1.66 GHz core2duo processor with 1024 KB RAM).
- The algorithm has been tested on some MCNC floorplan benchmarks and some randomly generated nonslicing floorplan. The tabulated result shows the circuits and the number of channel that it finds while portioning and the overall computation time.

Name of the floorplan (MCNC/ Randomly generated)	Number of staircase channels	Number of CPU cycle
Xerox	9 # of Level 0 channel -1 # of Level 1 channel -2 # of Level 2 channel -4 # of Level 4 channel -2	281
hp	Same as Xerox	357
n10	do	320
Randomly generated nonslicing floorplan nsl1	5 # of Level 0 channel -1 # of Level 1 channel -2 # of Level 2 channel -1	156
Randomly generated nonslicing floorplan nsl2	8 # of Level 0 channel -1 # of Level 1 channel -2 # of Level 2 channel -4 # of Level 4 channel -1	250

- Exact timing analysis is not done here, so it would not be clear whether the nets meeting their delay criterion.

- Time complexity: In the overall processes a considerable amount of time spent on finding the floorplan graph. As this procedure involves sorting and exhaustive searching. And all other are linear time procedure.

  The recurrence relation that it follows is :

  $T(n) = 2T(n/2) + O(n \log n) + O(n^2) + O(n)$

  $O(n^2)$ is for exhaustive search of the floorplan.

  $O(n \log n)$ is for sorting of rectangular blocks.

  $O(n)$ is for others.

This gives a solution of $t(n)=O(n(\log n)^2 + n^2)$.

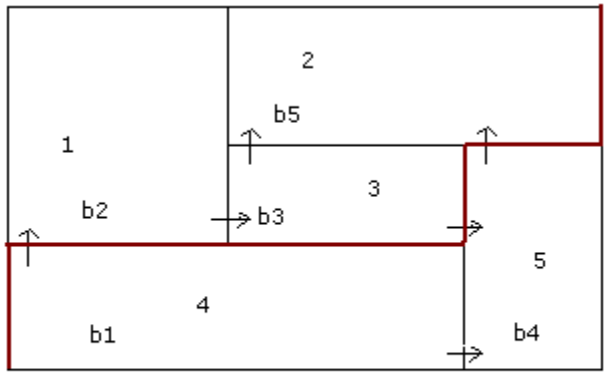
Scope of improvement:

Solution to the recurrence relation has the term $O(n^2)$ terms appears because of the exhaustive searching technique is used for finding adjacency of rectangular blocks. If we can reduce the searching to $O(n)$, the overall complexity will be reduced to $O(n(\log n)^2 + n\log n) \approx O(n(\log n)^2)$. Further improvement is possible provided floorplan graph can be generated in $O(n)$ then overall time complexity would reduced to $O(n \log n)$.

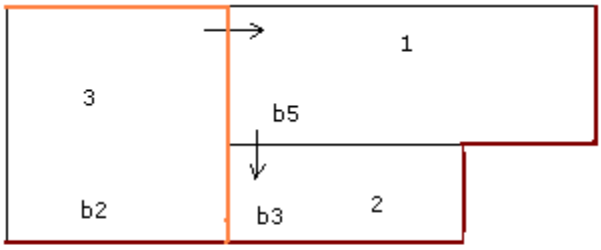
Future scope:

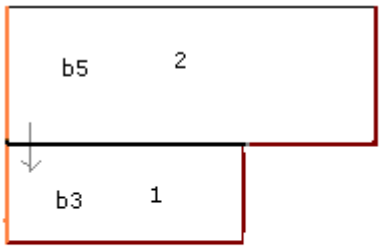
The output of the procedure reveals an overall connectivity of terminals of different. This information can be utilized to find the probable locations of repeaters for each net.

A worked out example of recursive staircase bipartitioning and the minimized cross-over connectivity for each channel is given here: A small nonslicing floorplan with 5 different nets is taken.

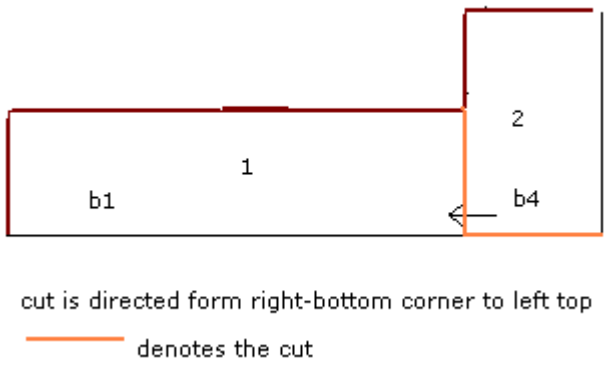
<table>
<tr>
<td> 

cut is directed form left bottom to right top
denotes the path</td>
<td>pins that are connected with their location :

cross over:0
connected
net 0 0.4 1.0 1.2 1.0
net 1 0.2 1.0 0.0 0.2
net 2 0.8 1.0 1.8 1.0
not connected
net 3
net 4</td>
</tr>
<tr>
<td> 

cut is directed form left top to right bottom
denotes the cut</td>
<td>cross over:2
connected
net 0 1.0 1.6 0.4 3.0
net 1 1.0 1.2 1.0 1.6
net 2 1.0 2.8 0.6 3.0
net 3 1.0 1.4 1.0 2.0
net 4 1.0 1.8 1.0 2.8
not connected</td>
</tr>
<tr>
<td> 

cut is directed from right top corner to left-bottom corner
denotes the cut</td>
<td>cross over:1
connected
net 1 1.8 2.0 1.6 2.0
net 2 2.8 2.0 4.0 2.8
net 3 2.6 2.0 3.0 2.0
net 4 1.6 2.0 2.0 2.0
not connected
net 0</td>
</tr>
</table>



cut is directed form right-bottom corner to left top

denotes the cut

cross over:0
connected
net 1 3.0 0.4 3.0 0.40
net 2 3.0 0.2 3.2 0.0
net 3 3.0 0.8 3.0 0.8
not connected
net 4

Fig 5.1 shows different paths and the pins belonging to the nets that are connected.

Chapter 6

Conclusion:

Here we presented an approach for recursive partitioning of floorplan and simultaneous minimization of crossover of nets. A simple heuristics for minimizing the net crossover is proposed here. The proposed approach has certain advantages (i) a *safe routing order* is always achieved for both nonslicing and slicing floorplan. (ii) Staircase path is targeted to partition a floorplan in almost equal halves with respect to the number of blocks in each level of recursion; the depth of hierarchy tends to be smallest. (iii) Since at level of hierarchical channel minimum number of pin is connected that simplifies the routing. Though overall time complexity is on the higher side we are investigating ways of reducing running time.

References

[1] W. C. Elmore, "The transient response of damped linear networks with particular regard to wide-band amplifiers," *J. Appl. Phys.*, vol. 19, pp.55–63, Jan. 1948.

[2] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*. Reading, MA: Addison-Wesley, 1990.

[3] R. Otten, "Global wires harmful?," in *Proc. Int. Symp. Physical Design*, Apr. 1998, pp. 104–109.

[4] J. Cong and D. Z. Pan, "Interconnect delay estimation models for synthesis and design planning," in *Proc. Asia South Pacific Design Automation Conf.*, Jan. 1999, pp. 97–100.

[5] M. Kang, W. W.-M. Dai, T. Dillinger, and D. LaPotin, "Delay bounded buffered tree for timing driven floorplanning," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1997, pp. 707–712.

[6] H.-M. Chen, H. Zhou, F. Y. Young, D. F.Wong, H. H. Yang, and N. sherwani, "Integrated floorplanning and interconnect planning," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1999, pp. 354–357.

[7] P.S. Dasgupta, P. Pan, S.C. Nandy, B. B. Bhattacharya, "Monotone bipartitioning problem in a planner point set with application to VLSI" in *ACM Transaction on Design of Electronic System, April 2002, vol. 7, issue 2,*pp. 231-248.

[8] SHERWANI, N. 1999. *Algorithms for VLSI Physical Design Automation*, 3rd ed. Kluwer Academic Publishers, Boston, MA.