STONY BROOK UNIVERSITY

Department
of
Electrical and Computer Engineering

ESE-556

VLSI Physical and Logic Design Automation

PROJECT- 3

GLOBAL AND DETAILED ROUTING

Submitted by-
NITIN ASTHANA
SBU ID - 12995559
nitin.asthana@stonybrook.edu

Project-3: Global and Detailed Routing

NITIN ASTHANA
SBU ID - 12995559
Graduate student
nitin.asthana@stonybrook.edu
Department of Electrical and Computer Engineering
Stony Brook University, Stony Brook, New York, 11790, US

Problem Statement- Develop, implement and experiment Soukup's algorithm for global routing and the net merger, detailed routing algorithm. The characteristics of the two algorithms are the following:

- Use placement information from your previous project 2 as input for detailed routing.
- The description of the channel is given in an input file. You design the format of the input file.
- The algorithm should be tested for various number of nets per channel as well as various number of terminals per net.
- Consider sufficiently many test cases to study the performance of the algorithm in terms of execution time and minimum number of tracks.

INTRODUCTION

Routing is choosing a path for a wire in network or between multiple networks. Routing is one of the most important steps in the VLSI physical design process. It basically involves generating metal wires for pins of same signals to connect with each other and also make sure that manufacturing design rules are obeyed. But before routing is done, the placement of a cell in a chip must be done. The actual physical connections are done in this process. While doing routing, considerations like routing channel capacities, wire widths and crossing has to be taken care of. The main objective of routing is to minimize the wire length plus numbers of vias such that each net meets its timing budget. We will be discussing Soukup's algorithm for global routing and the net merger, detailed routing algorithm in the further sections of the report. Global routing is one in which we generate a "loose" route for each net. We assign a list of routing region to each net such that we do not specify the actual layout of the wires. Detailed routing on the other hand is one in which we find the actual geometry of the layout of each net within the assigned routing regions.

RELATED WORK

In global routing one of the related work is done by Lee. Lee algorithm is one of the most famous algorithms for maze routing problems. It is used to find the shortest part in the maze and is used in routing. The algorithm uses Breadth First Search and uses queues to store the steps involving a queue to store the visited cells and a queue to store the neighbouring cells. The cells which are visited are removed from the queue and the process is continues until destination is reached.

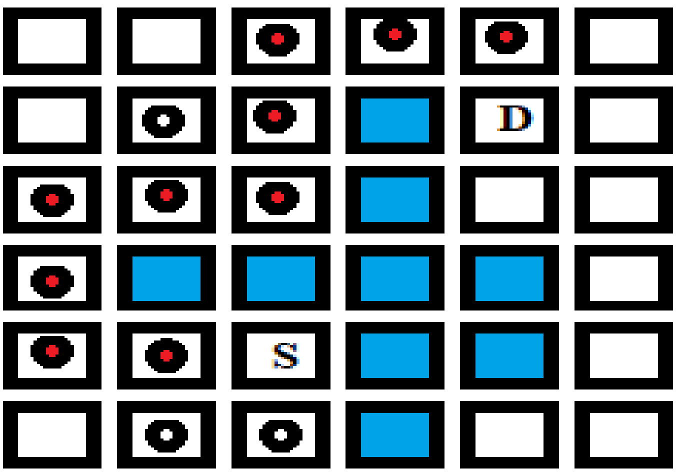
In detailed routing, greedy channel router is also one of the similar routing algorithms. The difference in this is that it routes the channel column by column starting from the left. This algorithm is able to handle the problems with cycles in VCG properly.

PROPOSED SOLUTION

Soukup's Algorithm

In global routing we will discuss the Soukup's algorithm. In lee Algorithm we saw that grid searches symmetrically to reach the destination. But Soukup's algorithm is a faster version of that algorithm using BFS (breadth first search) and DFS (depth first search).

To understand more properly let us use the help of the diagram shown below. Here "S" represents the source and "D" represents the destination and we need to find the shortest path between source and destination.



The DFS (or line search) will direct towards the destination D (as shown in the red circle). The DFS will continue until an obstacle or D is reached.

The BFS will be used similar to the LEE algorithm and will be used to bubble around the obstacle if an obstacle is reached. (In the figure above the empty circle represents the use of BFS).

Implementation of the Algorithm

I have initialised the chip area as 2D n*n matrix. There are two arrays dx[4] and dy[4] are also initialised with four values i.e. -1,0,1,0 which is used to travel in all the four direction.

Following function are used in the algorithm:

1) int targetDistance(int curr, int target): This function is used to calculate the between the source and destination. Here source means the cell at which you are standing right now, and it calculates the absolute distance between current location and the target.

2) int toDirectionDistance(int next, bool visit[], int target): This function will calculate the distance from the next cell which is unvisited and its distance to the direct. The function uses normal if conditions.

3) int neighbourDist(int curr, bool visit[], int des): This function will first select the neighbour which is closest to the target from the current location then will return its distance from the target.

4) bool SoukupAlgorithm(int x1, int y1, int x2, int y2):

- This is Boolean function which is used to tell whether there is path which exists from the current cell having the coordinates x1 and y1 to the target cell having the coordinates x2 and y2.
- I have used a stack plist which is used keep the track of the vertices and queue nlist which is used to keep the track of the neighbour vertices.
- The stack is initialised with the starting source and it marked as visited. The function continue until the stack is empty.
- I have used the normal DFS and BFS in this function.
- We first check recursively move from one cell to its neighbouring cell, until the current cell distance is equal to target distance.
- This is done in while loop. If we are not able to reach target and find any block in between then BFS is used and then we change the direction and move from current cell to adjacent cell.

- This is also done in a while loop. Towards the end of this function we are able to know whether there is a path from source to the target or not.

5) int main( ): This is a main function. In the main function I first read the input file which is a .txt file. The chip is initialised with “ 0 ”. The number of obstacles are arranged randomly using “ # ”. Then after reading all the inputs from the input.txt file, Soukup’s algorithm is processed. If the there a path exists from the source to the destination, the path is shown using “ * ” from source is marked as “S” and to destination “T”. If the path does not exist from source to destination, then source is represented using small “s” and destination using small “t”.

INPUT FILE

10 // it represents n which is the size of the chip area. 10*10 grid

9 // number of obstacles.

2 3 // location is obstacles

3 3 // ----“----

4 3 // ----“----

5 3 // ----“----

6 3 // ----“----

3 4 // ----“----

3 5 // ----“----

8 8 // ----“----

9 8 // ----“----

3 // number of targets for which we need to find the path

3 1 5 5 // the first two digits represents the x and y coordinates of the source

0 8 7 8 // the last two represents the coordinate of destination. Eg. S (3,1) D (5,5)

6 7 2 9

OUTPUT FILE

Soukup Algorithm:

Net (3,1)->(5,5) can be routed.

Net (0,8)->(7,8) can be routed.

Net (6,7)->(2,9) cannot be routed.

0 0 0 0 0 0 0 0 S 0

0 0 0 0 0 0 0 0 * 0

0 0 0 # 0 0 0 0 * t

0 S 0 # # # 0 0 * 0

0 * 0 # 0 0 0 0 * 0

0 * * # * T 0 0 * 0

0 0 * # * 0 0 s * 0

0 0 * * * 0 0 0 T 0

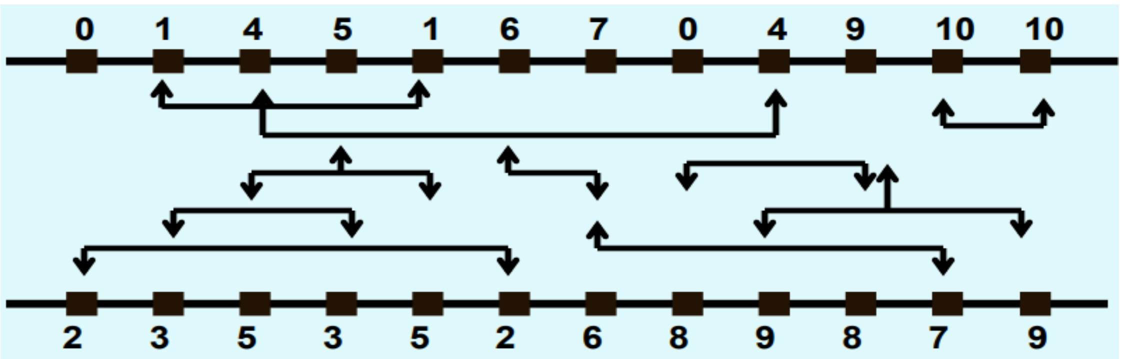
0 0 0 0 0 0 0 0 # 0

0 0 0 0 0 0 0 0 # 0

Since I was not able to implement the project 2 successfully, I have given my own input files.

Net Merger

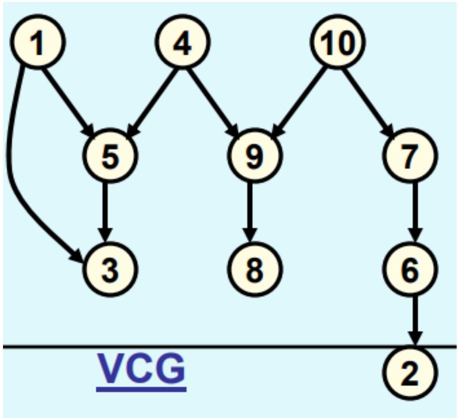
In detailed routing we will discuss the net merger routing algorithm. The main goal of detail routing is reduction of overall area by reducing the channel width. Let us understand the algorithm by using the following example.



Our first step will be to do the zone representation of the above connections.

For that we will first put all the nets into two rows arrays i.e top row and bottom row. Then I have created a function netsMap which have both the arrays.

Next, I have created a map which will arrange the connection which are given in the above examples i.e. void arrangeLines().

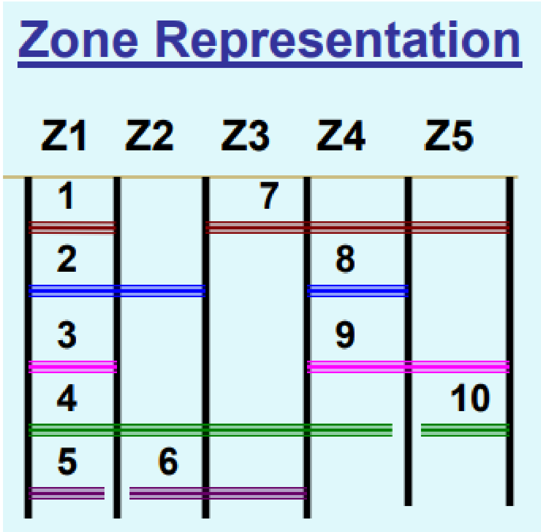


Our next step will be to make the VCG. To do this first we have to find the number of nets which are there in each column. To do that I have made a function NetsperCoulum(). After that I have created a function to make VCG. Basically, in this function I have stored the connections in a map.

Zone Table

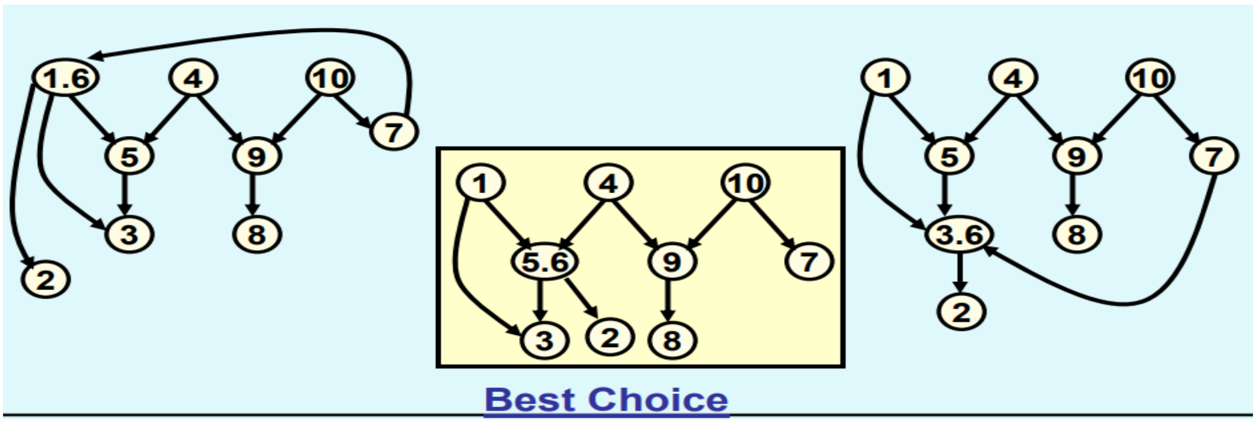
Column	S(i)	Zone
1	{2}	
2	{1,2,3}	
3	{1,2,3,4,5}	1
4	{1,2,3,4,5}	
5	{1,2,4,5}	
6	{2,4,6}	2
7	{4,6,7}	3
8	{4,7,8}	
9	{4,7,8,9}	4
10	{7,8,9}	
11	{7,9,10}	5
12	{9,10}	

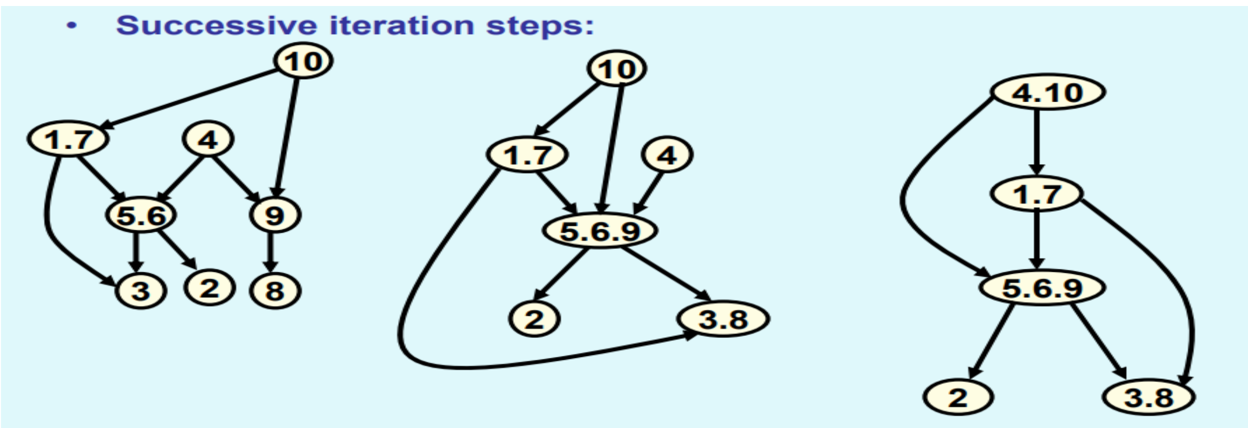
Now I have to create a zone table as shown in the above figure. To do that I have create a function makeZone(). But we have to make sure that the zone number should be started from number 1. With the help of the NetsperCoulum() function I have made zone table and stored the zones in a vector.



To make the zone representation I have make zoneRepresentation() function and have stored the the information a map netZoneRep.

Till here I was able to implement the algorithm successfully. After that our algorithms requires to merge the nets iteratively.





The merging of these nets I was not able to implement successfully. Assuming if the nets are merged like in the last figure last diagram. The next step is the track assignment. Track assignment requires each node to be assigned separate track. To do that we can use left-edge algorithm to assign horizontal tracks. Apparently, I implementation of that part was also not done. Manually if one does, the list of nets sorted on their left edges, subject to the vertical constraint will be:

[4-10, 1-7, 5-6-9, 2, 3-8]

Track 1: Nets 4 and 10

Track 2: Nets 1 and 7

Track 3: Nets 5, 6 and 9

Track 4: Net 2

Track 5: Nets 3 and 8.

I have attached the functions which I was able to implement in the appendix.

IMPLEMENTATION ISSUES

In Soukup's algorithm one of the major issues I was having was how to change the directions and jump to adjacent cells and start the algorithm from that cell. To solve this I used dx[4] = { -1,0,1,0 }, dy[4] = { 0,-1,0,1 } and x = nx + dx[i], y = ny + dy[i]. Also, since I was not able to implement the project 2 it was difficult to start and decide the initial input for the algorithm.

In Net merger, there are a lot of hashmaps which are made. Plus, all the maps have one of the variables as vector. Therefore, I have to keep a track of a lot of maps and a lot of vectors at the same time. As a result of this the running time of the algorithm is increasing a lot and a lot of confusion was happening. As a result of which I was not able to implement the merging of net function and assigning the track and track number function properly. Time constraint was also there otherwise I would have tried more to get the output. I was trying to merge the nets solution sequentially as a result of which optimal merging was not happening and it was stopping at an earlier stage then required. A more proper way to implement the algorithm is required.

RESULT

Though I did not have the output of the placement of the cells in from project 2 using the IBM files, but I was successfully able to implement the Soukup's algorithm for global routing. I have shown the input and output of one case in the implementation. I have tried the implementation with other and large input files as well, it was running successfully. Therefore, I assume, if I will have the coordinates of the cell of the IBM test benchmarks after placement strategy, my implementation will be successful in doing the global routing.

Next, for Net Merger algorithm I could not implement the algorithm completely. Therefore, I cannot compare and talk about the results.

CONCLUSION

Soukup's algorithm is an efficient algorithm in performing the global routing. It is able to achieve better results in comparison to the Lee's algorithm in performing the global routing using BFS and DFS. I can conclude by saying I was able to implement the Soukup's Algorithm successfully.

Net Merger is detailed routing algorithm which does not allow doglegs or cycles in the VCG. It is one of the most efficient detailed routing algorithms i.e. rather than assigning different nets to different tracks multiple nets share same track. I have tried to explain the algorithm with help of an example, but I could not implement the Net merger algorithm successfully.

BIBLIOGRAPHY

[1] Detailed Routing (Accessed on May 12th, 2020) <https://www.facweb.iitkgp.ac.in/~isg/CAD/SLIDES/12-detailed-routing.pdf>.

[2] VLSI Routing by Naveen Kumar (Accessed on May 14th, 2020) <https://www.slideshare.net/NaveenKumar11/vlsi-routing>

[3] VLSI Physical design automation - David Pan

[4] An efficient approach for four-layer channel routing in VLSI design – Ajoy Kumar Khan and Bhaskar Das

[5] Maze router (Accessed on May 12th, 2020) <http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf>

APPENDIX

Soukup's Algorithm

```
#include <cstdlib>
#include <stack>
#include <queue>
#include <climits>
#include <cstring>
#include <bits/stdc++.h>

using namespace std;
char chipArea[100][100];
int n, nblocks, bx, by, numtarget, sx, sy, desx, desy;
bool Sblock[10000];
int Spath[10000], dx[4] = { -1,0,1,0 }, dy[4] = { 0,-1,0,1 };  //this is used to move in 4 directions
int srcX[100], srcY[100], desX[100], desY[100];
int super_id;

int targetDistance(int curr, int target) {
    int dis;
    int tx = target / n, ty = target % n, cx = curr / n, cy = curr % n;
    dis = abs(tx - cx) + abs(ty - cy);  //calculating the absolute distance
    return dis;
}

int toDirectionDistance(int next, bool visit[], int target) {
    int dis = INT_MAX;
    int tx = target / n, ty = target % n, nx = next / n, ny = next % n;

    for (int i = 0; i < 4; i++) {
        int x = nx + dx[i], y = ny + dy[i];  //used to change the direction and jump to adjacent cells
        int id = x * n + y;
        if ((x >= 0 && x < n && y >= 0 && y < n) && visit[id] == false && Sblock[id] == false) {
            if (dis > abs(tx - x) + abs(ty - y)) {
                dis = abs(tx - x) + abs(ty - y);
                super_id = id;
```

```
            }
        }
    }
    return dis;
}

int neighbourDist(int curr, bool visit[], int des) {
    int x = curr / n, y = curr % n;
    bool flag = true;
    int d;

    if (toDirectionDistance(curr, visit, des) <= targetDistance(curr, des)) {
        flag = false;
        d = super_id;
    }
    return flag == true ? -1 : d;
}

bool SoukupAlgorithm(int x1, int y1, int x2, int y2) {
    bool visit[n*n];
    memset(visit, false, (n * n) + 1);
    int src = x1 * n + y1, des = x2 * n + y2;
    stack<int>plist;
    queue<int>nlist;
    //cout << "in plist " << src << endl;
    plist.push(src);
    Spath[src] = -1;
    visit[src] = true;

    while (!plist.empty()) {
        int pid = plist.top();

        if (pid == des) {
```

```
            return true;
        }

        if (toDirectionDistance(pid, visit, des) <= targetDistance(pid, des)) {
            int id = super_id;
            plist.push(id);
            visit[id] = true;
            Spath[id] = pid;
            if (id == des) {
                return true;
            }

            while (neighbourDist(id, visit, des) >= 0) {
                int new_id = neighbourDist(id, visit, des);
                plist.push(new_id);
                visit[new_id] = true;
                Spath[new_id] = id;
                if (new_id == des) {
                    return true;
                }
                id = new_id;
            }
        }
        while (!plist.empty()) {
            pid = plist.top();
            int tx = pid / n, ty = pid % n;
            for (int i = 0; i < 4; i++) {
                int x = tx + dx[i], y = ty + dy[i];
                int id = x * n + y;
                if ((x >= 0 && x < n && y >= 0 && y < n) && visit[id] == false && Sblock[id] == false) {
                    nlist.push(id);
                    visit[id] = true;
                    Spath[id] = pid;
```

```cpp
                }
            }
            plist.pop();
        }
        while (!nlist.empty()) {
            plist.push(nlist.front());
            nlist.pop();
        }
    }
    return false;
}

int main()
{
    ifstream in("input.txt");
    ofstream out2("soukup.txt");
    in >> n;
    in >> nblocks;
    cout << "n" << n << endl;
    cout << "nblocks" << nblocks << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            chipArea[i][j] = '0';
        }
    }
    memset(Sblock, false, (n * n) + 1);
    for (int i = 0; i < nblocks; i++) {
        in >> bx >> by;
        chipArea[bx][by] = '#';
        Sblock[bx * n + by] = true;
    }
    bool ans[2][100];
    in >> numtarget;
```

```
cout << "numtarget" << numtarget << endl;
for (int i = 0; i < numtarget; i++) {
    in >> sx >> sy >> desx >> desy;
    srcX[i] = sx;
    srcY[i] = sy;
    desX[i] = desx;
    desY[i] = desy;
    chipArea[sx][sy] = 'S';
    chipArea[desx][desy] = 'T';
}
//input reading successfully
out2 << "Soukup Algorithm : " << endl;
int i = 0;
while (i < numtarget) {
    sx = srcX[i];
    sy = srcY[i];
    desx = desX[i];
    desy = desY[i];
    if (SoukupAlgorithm(sx, sy, desx, desy) == true) {    //printing of the output files
        int id = desx * n + desy;
        Sblock[id] = true;
        Sblock[sx * n + sy] = true;
        while (Spath[id] != -1) {
            chipArea[Spath[id] / n][Spath[id] % n] = '*';
            Sblock[Spath[id]] = true;
            id = Spath[id];
        }
        chipArea[sx][sy] = 'S';
        ans[1][i] = true;
        if (ans[1][i] == true) {
            out2 << "Net (" << sx << "," << sy << ")->(" << desx << "," << desy << ") can be routed." << endl;
        }
    }
```

```cpp
        }
        else {      //if there is no route
            chipArea[sx][sy] = 's';
            chipArea[desx][desy] = 't';
            ans[1][i] = false;
            for (int j = 0; j < numtarget; j++) {
                if (ans[1][j] == false) {
                    out2 << "Net (" << sx << "," << sy << ")->(" << desx << "," << desy << ") cannot be routed." << endl;
                }
            }
            out2 << endl;
        }
        i++;
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) out2 << chipArea[i][j] << " ";
        out2 << endl;
    }
    out2 << endl;
    return 0;
}
```

Net Merger Algorithm

```
#include <iostream>
#include <map>
#include <fstream>
#include <list>
#include <vector>
#include <algorithm>
using namespace std;

map<int, list<int>> netMap;
map<int, border> linesMap;
map<int, vector<int>> columnNets;
map<int, vector<int>> VCG;
map<int, vector<int>> veczoneTable;
map<int, vector<int>> netZoneRep;

struct lines
{
    int minimum;
    int maximum;
    int netNo;
    int netStart;
    int netEnd;
    int track = -1;
};

void netsMap()
{
    int itr = 0;
    while( itr = 0; itr < topRow.size())
    {
        netMap[topRow[itr1]].push_back(itr1);
        netMap[bottomRow[itr1]].push_back(itr1);
        ++itr;
```

```
    }
}

void arrangeLines()
{
    map<int, list<int>>::iterator itr;
    int value = 0;
    itr = netMap.begin();
    while( itr != netMap.end())
    {
        lines l;
        itr->second.sort();
        b.netNo = val;
        b.minimum = itr->second.front();
        b.maximum = itr->second.back();
        linesmap.insert(pair<int, lines>(value,l));
        value++;
        itr++
    }
}

void NetsperCoulum()
{
    map<int, lines>::iterator itr1;
    int itr = 0;
    while ( itr < topRow.size())
    {
        itr1 = linesmap.begin();
        while ( itr1 != linesmap.end())
        {
            if (itr1->first != 0)
            {
```

```
                if (itr1->second.min <= itr && itr <= itr1->second.max)
                {
                    columnNets[itr].push_back(itr1->first);
                }
            }
            ++itr1;
        }
        ++itr;
    }
}

void makeVCG()
{
    int itr = 0;
    while (itr != topRow.size(); )
    {
        if(topRow[itr] != 0)
        {
            VCG[topRow[itr]].push_back(bottomRow[itrV]);
        }
        ++itr;
    }
}

void makeZone()
{
    int zoneNumber = 1;
    vector<int> temp;
    itr = columnNets.begin();
    temp = itr->second;
    itr = columnNets.begin();
    while ( itr != columnNets.end())
```

```
    {
        if (includes(itr->second.begin(),itr->second.end(),temp.begin(),temp.end()))
        {
            temp = itr->second;
            veczoneTable[zoneNumber] = temp;
        }
        else if (includes(temp.begin(),temp.end(),itr->second.begin(),itr->second.end()))
        {
            veczoneTable[zoneNumber] = temp;
        }
        else
        {
            zoneNumber++;
            temp = itr->second;
            veczoneTable[zoneNumber] = temp;
        }
        ++itr;
    }
}
void zoneRepresentation()
{
    map<int, vector<int>>::iterator itr;
    vector<int> :: iterator itrV;
    itr = zoneTable.begin();
    while ( itr != zoneTable.end())
    {
        for(itrV = itr->second.begin(); itrV != itr->second.end(); ++itrV)
        {
            netZoneRep[*itrV].push_back(itr->first);
        }
        ++itr;
    }
}
```