EE215A VLSI Design Automation

Spring 2023

Project: Router

Due date: 23:59, June 2, 2023

In lecture, we described how to use the maze routing method to route the wires in a large ASIC, using a 3-dimensional stack of routing grids. In this project assignment, you get to build a version of a real maze-router and run on it some synthetic benchmarks (to test individual routing features) and on some real industrial benchmarks. The assignment has two parts:

- Phase I: Build a core router. This will handle 2-point nets, non-unit costs in the routing grid, bend penalties, and 2 separate routing layers. You route the nets in the order they appear in the input file. There are two input files: one describes the geometry of the routing grids (.grid file); the other describes the netlist of wires to connect (.nl file). We will give you a series of increasingly complex benchmarks that test specific features of the router.
- Phase II: We will give you some bigger industrial benchmarks to play with. You can try your final router on these, and see what happens. Even if your router does not implement all the functionality of the most complex router, you can probably route a lot of nets in these designs, and acquire some additional points.

You can write this program in C/C++/Python. We will provide inputs as a pair of simple textfiles: a netlist file (which tells you what nets to route, and where their pins are) and grid file (while tells you what the routing grid looks like, where the obstacles and non-uniform cells costs are). You will output your final routed result as a simple textfile in a fixed format.

1 Core Router

The nice thing about maze routing is that the "serious core" of the algorithm is not very complex, when implemented as software. And, one can add features in an incremental way, by starting with a simpler router, and adding code to add these features. So, we suggest you follow a strategy where you first build and test the most basic features, before you add more complicated features.

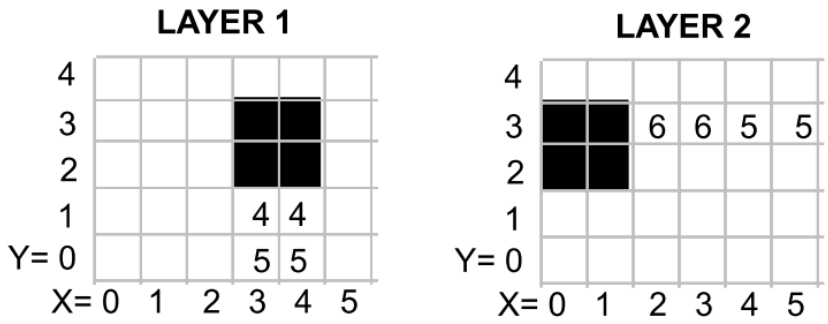
1.0.1 Input File Format

You need to know two things to do this router. First, you need to know what the routing grids look like, so you can search for paths using the maze routing method from lecture. Second, you need to know what the netlist is – what wires we want to route, where their pins are. So, we use simple text formats to specify each placement problem. Your router will read two input files: the netlist file (which wires to route, where are their pins) and the grid file (describes the routing grids).

Grid File: The grid file specifies the physical grid on which your maze router will search. It has this format:

- First line: four integers: X grid size (columns), Y grid size (rows), the Bend penalty (for use in maze search) and the Via penalty (also for use in maze search)
- The next $X \times Y$ integers in the file: specify the costs associated with each grid cell in LAYER1 of the routing. Most cells will be specified with a 1: these are free to route on, and they have unit cost. If a cell has a non-uniform cost, it will appear as a positive number. If the cell is blocked (you cannot use it to route), it will appear as a negative number, specifically a -1.
- The final $X \times Y$ integers in the file: specify the costs associated with each grid in the LAYER2 of the routing, using the same rules as LAYER1.
- Grid cell order: We specify the grids in coordinate order: x coordinates from 0 to $X-1$; y coordinates from 0 to $Y-1$.

Here is a small example with the grids drawn in detail, and the associated input grid file shown. A white cell is available to route and has cost=1. A black cell is an obstacle, unavailable to route. A small positive integer shows a free cell with nonuniform routing cost.



Here is a detailed text file input for these grids. Note: the comments are for illustration only, we do not put any comments in your input grid file.

```
//EXAMPLE GRID FILE FORMAT FOR ABOVE GRID DIAGRAM
6 5   5 10      // X gridsize  Y gridsize  BendPenalty  ViaPenalty
1 1 1 5 5 1    // 1st row of LAYER1, where y=0
1 1 1 4 4 1    // 2nd row of LAYER1, where y=1
1 1 1 -1 -1 1  // 3rd row of LAYER1, where y=2
1 1 1 -1 -1 1  // 4th row of LAYER1, where y=3
1 1 1 1 1 1    // 5th row of LAYER1, where y=4
1 1 1 1 1 1    // 1st row of LAYER2, where y=0
1 1 1 1 1 1    // 2nd row of LAYER2, where y=1
-1 -1 1 1 1 1  // 3rd row of LAYER2, where y=2
-1 -1 6 6 5 5  // 4th row of LAYER2, where y=3
1 1 1 1 1 1    // 5th row of LAYER2, where y=4
```

Netlist File: The netlist file specifies the nets you need to route. Here are the constraints on our nets, for this project:

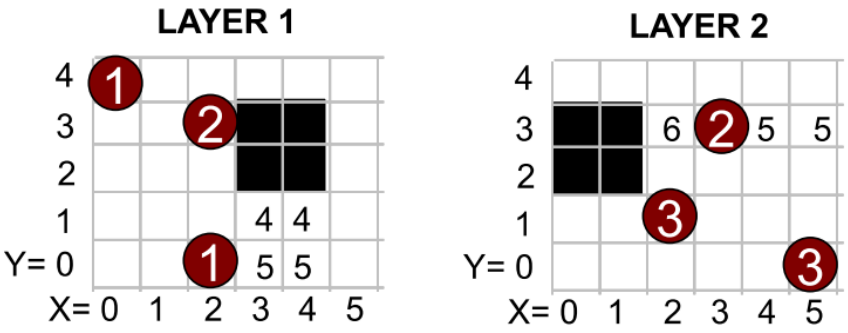
- All have just 2 pins. So, these are "2 point" nets. This simplifies the programming and the grading.
- We specify a NetID for each net, which is just an integer. We specify the nets in NetID order in this input file: 1, 2, 3.. etc
- We specify an (X,Y) coordinate for each pin on each net.
- We specific a layer L=1 or 2, for each pin on each net.
- You route the nets in NetID order, i.e., in the order we specify the nets in this input file.

The file format itself is simple:

- The first line: one integer: NetNumber, which is how many nets in this problem.
- The next NetNumber lines: each specifies one net to route.
- To specify each net: Each net is specified with five integers:

NetID LayerPin1 Xpin1 Ypin1 LayerPin2 Xpin1 Ypin

So, we tell you the name of the net (NetID) and for each pin (1, 2) the layer, and the (X, Y) coordinates. As an example, here is the same set of grids, with three nets specified. We have circles with numbers $1, 2, 3$ to identify the pins that specify each net.



Here is the netlist input file that corresponds to these nets to be routed. Again, the comments are for explanation only; we do not add comments to the actual file:

```
//EXAMPLE NETLIST FILE FORMAT FOR ABOVE GRID DIAGRAM
3       // NetNumber = 3 nets to be routed
1  1 0 4  1 2 0   // Net 1.  Pin1 Layer1 (0,4);  Pin2 Layer1 (2,0)
2  1 2 3  2 3 3   // Net 2.  Pin1 Layer1 (2,3);  Pin2 Layer2 (3,3)
3  2 2 1  2 5 0   // Net 3.  Pin1 Layer2 (2,1);  Pin2 Layer2 (5,0)
```

1.0.2 Output File Format

Your router will find a path for each net, using the cells in the grids and maybe some vias. Or, you might not be able to route a net. Your output tells us what happened with each net you tried to route, and what path it took.

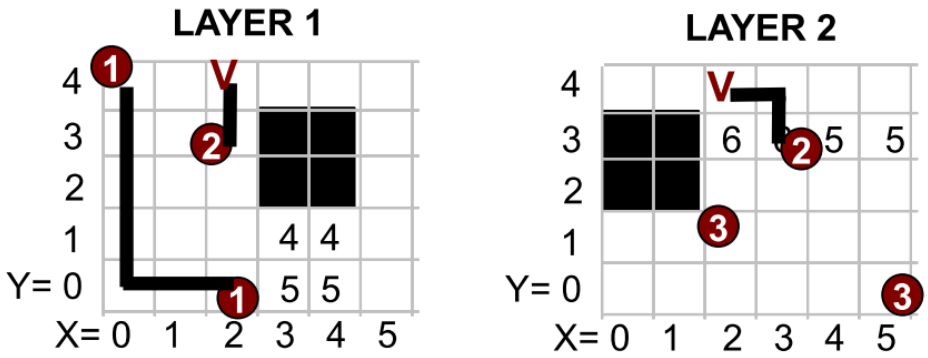
This is also a simple text file, with the following format:

- First line: One integer, NetNumber, how many nets are in this benchmark. This is just the same as in the input netlist file.
- For each net in order 1, 2, ... NetNumber: describe the path each net took.
  - First line: NetID (an integer)
  - Next lines: Describe each cell in the routing path, one cell per line of this file. Each line is 3 integers: LayerInfo Xcoord Ycoord.
  - LayerInfo: is 0, 1, 2 or 3. If your cell is on Layer1, this is 1; if your cell on Layer2, this is 2. If this is a via, put a 3. To signal the end of this list, put a 0 on the line with nothing else.

Note that you indicate a net that you could not route by just have a single line after the NetID with a 0.

Here is our small example again, now with a real path drawn for each. Note the following:

- Net 1: is routed entirely on Layer1.
- Net 2: is routed on both layers, and it has a via at (2,4), indicated in the grid with a "V" on each layer.
- Net 3: is unrouted. This is just for illustration purposes.



The basic idea is you just list every cell on the path, in order, from the first pin listed in the netlist file, to the second pin listed in the netlist file. More precisely, we expect you to start routing with the first pin in the netlist file as the source, and to route to the second pin as the target. List the path in order, from source cell to target cell.

A few subtle points here, to be clear about:

- Make sure you list every cell on the path.
- List the starting source cell as the first cell on the path.
- If there is a via at location (X,Y), you will create 3 lines in the output file: you got to cell (X,Y) on one layer, so you list this cell on this layer; you put a via at (X,Y), so you list this via; you expanded to the other layer at cell (X,Y), so you list this cell at this other layer, as well. This is very important for the visualization tool and the auto-grader to work properly.
- List the ending target cell as the last cell on your path.
- If you fail to route a path, you still have to list the net. But, you just put the "0" marker on a line, to indicate there are no cells on the path.

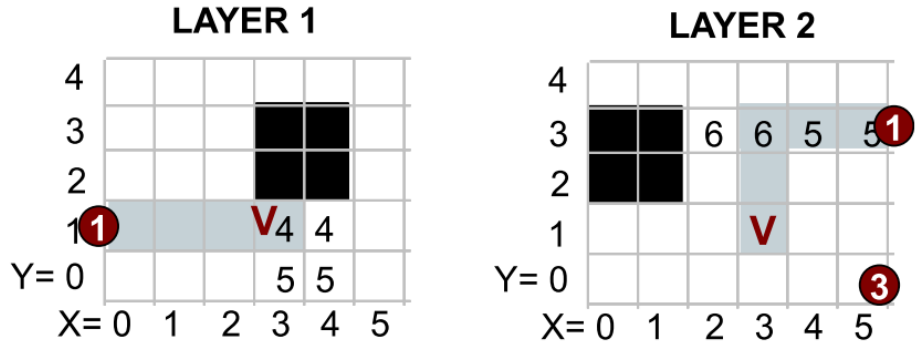
```
//EXAMPLE OUTPUT FILE FORMAT FOR ABOVE GRID DIAGRAM
3       // NetNumber = 3 nets we have tried to route
1       // Net 1 starts here
1 0 4   // First cell on Net1 path is Layer1 at (0,4)
1 0 3
1 0 2
1 0 1
1 0 0
1 1 0
1 2 0   // Last cell on Net1 path is Layer1 at (2,0)
0       // Net1 end
2       // Net 2 starts here
1 2 3   // First cell on Net2 path is Layer1 at (2,3)
1 2 4
3 2 4   // Net2 has a via at (2,4)
2 2 4   // Net2 starts again at (2,4) thru this via on Layer2
2 3 4
2 3 3   // Last cell on Net2
0       // Net2 end
3       // Net3 starts (remember, it was unrouted)
0       // Net3 end
```

1.0.3 Evaluating router results

We have three basic criteria for judging your router's results:

- Completion: Did you correctly route the net, yes or no?
- Cost: What was the total cost of the path you used to route this net?
- Shape: Did your router do the right thing, and correctly find a route that has the expected shape (such as avoiding obstacles, no zig-zag route when bending cost is high, etc.)?

It is worth noting that the calculation for cost means we add up the cost of all the cells and vias and bends on your routing path. concrete example of how this cost calculation works:



The diagram above shows the path (shaded) of Net1, which starts on Layer1 and ends on Layer2, and inserts one via at (4,1). The cost of this path would be computed as follows, cell by cell:

```
Starting on Layer1, from cell (0,1)
1 + 1 + 1 + 4        // 3 unit cost cells, and 1 non-unit cells
+ 10                 // the Via penalty from our Grid input file
+ 1 + 1 + 6          // cell costs on Layer2, going vertically
+ 5                  // Bend penalty to turn from North to East
+ 5 + 5              // final cells on horizontal path to target
-----------------------------------------------------------------
40 = Total path cost
```

2 PHASE-I: Core Router

2.1 Suggested Building Strategy

As we mentioned in the introduction, the nice thing about maze routers is you can build a "lot of" router with "a very little" code. So, we suggest you build the router incrementally, by starting with basic functionality, and then add more features. Our benchmarks are organized this way too. As you add features, you can try the more complex benchmarks. We suggest you build the router features in this order:

- One Layer Unit-Cost Router: this is the most simple router. You expand the source cell until you reach the target cell. Every cell has unit cost. There are no bend penalties. You route on just one layer. With this router, you can run our first benchmark, which has all of its nets on Layer1. So, you read in both grids (for Layer1 and Layer2) but you can ignore Layer2 entirely.

- One Layer Router with Bend Penalty and Non-Unit Cost: Starting from the previous router, add bend penalties, and non-unit costs from grid cells. Mechanics to deal with these two features are very similar: you have to add something "different" when you expand a non-unit cell, and you must pay attention to expansions that change directions, and add something else (the bend penalty). To keep things simple, you can stop when you reach the target cell (i.e., ignore other problems with the inconsistent cost function). With this router, you can run our second benchmark, which also has all its nets on Layer1.

- "No Vias" 2-Layer Router: Starting from the previous router, add a second routing layer. This router is "No Vias" because we never give you nets that have pins on two different layers. So, you will get some nets that can be routed entirely on Layer1, and some nets that can be routed entirely on Layer2. You do not need to implement the vias to route this benchmark. This really means just a few extra features. (1) When you add a cell to the heap, you need to remember what layer it was on. (2) When you print out the path for each wire, remember to print the right layer for each cell.
- Real 2-Layer Router: Starting from the previous router, add the vias, so you can change layers as needed in this router. This means just a few extra features.
    1. When you expand a cell, as well as north/south/east/west expansion directions, you have to try to expand it to the other layer, i.e., up from layer 1 to 2, or down from layer 2 to 1.
    2. When you expand through a via and change layers, you must remember to add in the via cost.
    3. When you print out the path for each wire, remember to print not only the right layer for each cell, but also print the vias as well.

2.2 Core Router Benchmark Details

There are 5 benchmarks related to the Core Router:

1. 1-Layer 2-Point Unit-Cost Routing: we have a simple synthetic benchmark with 20 nets with simple routing patterns, all in Layer 1. Straight lines, simple bends, etc.
2. 1-Layer 2-Point Bend-Penalty Non-Unit-Cost Routing: we have a simple synthetic benchmark with 20 nets in a grid with non-uniform costs. Again, we use simple routing patterns, all in Layer 1. Now, we look not only at your ability to connect each net, but the cost of each of your nets. We do this by adding up the cost of all the cells on your net: unit cost cells, non-unit-cost cells.
3. "No Via" 2-Layer 2-Point: This synthetic test has 16 nets, 8 nets in Layer1, 8 nets in Layer2, and you do not need to implement the vias. You just need to be able to read a netlist with nets on 2 different layers, and route them on these isolated layers.
4. Real 2-Layer 2-Point Router: This synthetic test has 15 nets and several of these nets have pins that require you to cross the layers. If you can't deal with the vias, you can't route this benchmark. (By the time you get to this benchmark, we assume you can do the file formats correctly.)
5. Industrial 2-Layer Design: This is a semi-real benchmark, You have 2 routing layers to use to route the nets. We have simplified the netlist, because the real benchmark has many nets with more than 2 pins; we have arbitrarily selected a pair of pins from a subset of these multi-pin nets. There are 128 nets in this benchmark. We do not expect you to be able to route all these net, but you should try your best to route as much nets as possible. This industrial benchmark has one special "constraint" you need to be aware of. We put blockages on all the pin locations, for this test case. The idea is simple: if we don't do this, an early net you route might block all the pins you need to use to start/end some future nets. So, the way you deal with is simple: when you route a new net, unblock the source and target pins as the first step. Also, we recommend you take a look at some of the ideas for the Phase II, since you might want to try those on this test case as well, if you are feeling ambitious.

In conclusion, for the first five benchmarks, you objective should be:

1. Output the valid format to make sure you have a correct format for our auto-grader.
2. Successfully routed the nets as much as possible and having reasonable costs.

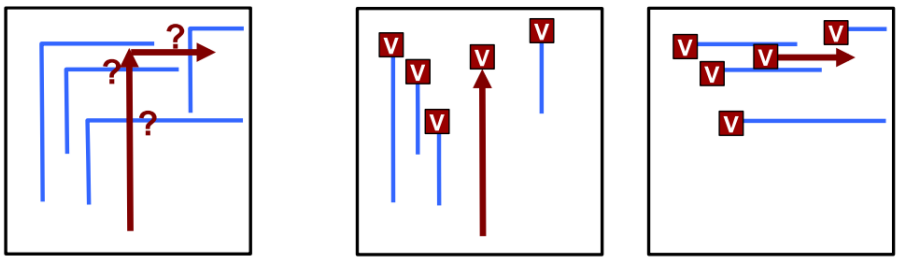
3 PHASE-II: Advanced Router

We will supply 3 more big industrial benchmarks for you to try to route. As with the core router, you need to be able to route in 2 layers with vias to attempt these. As with the industrial benchmark in the required part of the assignment, note that every source/target pin in the benchmark will be blocked on both Layer1 and Layer2 in the input. As a first step to routing these, for each new net, you need to unblock the source and target cells on both layers. You do not need to do any additional code to try these, you just need to be willing to run your router engines for longer (several minutes), to see what happens.

Once you have tried your basic algorithm to see how well it performs (as a baseline), you might want to consider trying the following modification, which is what real industrial-strength routers use. You can then compare the results of your basic router and your new one with the modification to see which performs better (number of nets routed, wire length, wire cost, etc).

3.1 Prefered Direction Routing

The way a real, industrial-strength router manages to route millions of nets on 8-12 layers of physical metal routing, is to impose preferred direction constraints on the layers. This is most commonly done by restricting alternating layers to be "mostly vertical" and "mostly horizontal". So, for us, we suggest you make Layer1 mostly-vertical routing, and Layer2 mostly horizontal. This means you will use a lot of vias. But this also you means you will not block wires on one layer with "wrong way" wires going the other direction. The idea is illustrated in the diagram below, where the red boxes with "v" indicate vias.



If we let wires go both horizontally and vertically on each layer – we create many obstacles ("?s" in diagram)

Layer1 is now **Mostly Vertical.** New (red) wire path less blocked on this layer.

You can accomplish this in one of two ways:

- Implement strictly one-way routing: This is the easiest to implement. We suggest you try this first. The idea is really easy. Layer1 is only allowed to expand cells in the vertical direction: north or south. You just disallow east and west expansions. Similarly, Layer2 is only allowed to expand cells in the horizontal direction: east or west. You just disallow north and south expansions. So, when a wire needs to find a path with both horizontal and vertical elements you must use a via – or a lot of vias. It turns out this makes routing lots of interacting wires.
- Implement preferred one-way routing: This is exactly like a bend penalty. You still allow "wrong way" expansion. But, you add a penalty. So, if you are on a preferred-vertical layer, you add a cost to each cell that wants to expand the path to the east or the west. This means you allow "short" wrong way wires, but not long wrong way wires? How short? You get to decide, by balancing the Wrong

way penalty, the Bend penalty, and the via costs. What is the right balance between these costs? Experiment, experiment, experiment! In the real world, people get paid lots of money to tune these parameters in industrial routers.

3.1.1 Advanced Router Benchmark Details

There are 3 benchmarks. All of the following benchmarks are 2-layer semi-real industrial designs, converted from placement results. All the nets are 2-point net.

- fract2: 125 nets. The scale of this benchmark is similar to the last benchmark in Phase I. We do not expect you to route all the nets.
- primary1: 830 nets. This is a complete industrial design, which contains a lot more nets than the previous benchmarks. Similarly, we do not expect you to route all of the nets, but a fraction of them.
- industry1: 1000 nets. This is the largest benchmark in this programming assignment. The original benchmark contains more than 2000 nets. We selected a subset of 1000 nets whose pins are distributed fairly even in the chip area.

4 Submission Details

You should submit following files, so we can evaluate the correctness and quality of your routing result.

- The source code, a build script if you are using the C++/C, a "requirements.txt" file if you are using Python to list the libraries your program depends on.
- A README.md to tell us how to run your program.
- The output routing result of each benchmark. The file name should be the same as the the benchmark name with ".route" extension. For example the output file name of "bench1" should be "bench1.route".
- A report to tell us what you have done, some implementation details , etc.
- All the source code/build scripts/README.md should be put into the "src" folder, all output files should be put into the "out" folder, the report should be put into "rpt" folder.

5 Grading Details

We will consider each benchmark separately, and rank each benchmark. The ranking criteria are as follows:

- The router that can successfully route more nets is ranked higher.
- If the number of successfully routed nets of two router is the same, We compare the sum of the cost of all routing path, and the lower cost router's rank is higher.

For each benchmark, we will get a normalized score based on your ranking. At the same time, we will assign a weight w according to the difficulty of each benchmark. We use the weighted sum of all benchmarks to calculate your final ranking. This scoring standard is just a reference, and we will make specific adjustments according to your completion. The overall goal of your router should be: first, route as many nets as possible, and then improve the routing quality as much as possible.