EE 541 – Automated Layout of Integrated Systems
Program #5: Global Routing

Given the placement you created in program #4, we now have to handle global routing. This will then be refined by the detailed routing done in the next assignment.

Algorithm Description

Your global router will be built from a simple maze routing algorithm. You start from the source of a net, and perform a breadth-first search of the routing graph, assigning distances to the locations you encounter. All routing regions have a length of “1”, making this easy. Once you reach a destination of the net, you must backtrack to find the path. That is, if you reached the destination with a distance of 5, then the route should connect from here back to a location at distance 4, which connects to one at distance 3, etc. until you reach the source.

If you are routing a multi-terminal net, once you have connected up the source with the first terminal, you must iterate on the rest of the terminals. However, to minimize wiring area, you should treat every point of this previous routing as a source of the maze algorithm. That is, every location of the existing routing becomes a starting point of the breadth-first search, with distance 0. Once you reach another terminal with the maze router, you backtrack to any point on the prior routing, and join up with it. This allows for some sharing of routing resources between different destinations. Iterate on this process until all terminals are reached.

Your programming efforts will focus on the function routeNet in GlobalRouter.java. This routine routes a single net in the circuit, using the maze algorithm discussed above. When you are done, you will have produced a global routing of this net. This is represented via the NetGlobalRouting data structure, which has one instance per routing region you routed through. In general, you will create these instances during the backtracking step described above for each destination terminal. You will also use the GRlocation structure, which represents specific routing regions. It is important to remember that GRlocations represent routing regions, while NetGlobalRoutings represent the metal routing a net through a routing region.

Getting Started

You will build upon the code you produced in the previous programming assignments. Again, to run the program you compile and run “CircuitViewer.java”. Then, open a circuit (File->open) and start placement (Edit->Place & Route Circuit). Note that to do global routing you do not need to do placement first – the initial placement produced, while far from optimal, can be the starting point for global routing. Thus, don’t bother running any placement algorithms during debugging.

Before you begin coding, you first need to understand how the routing regions are named. As shown in Figure 1 there are three types of routing regions: TERMINAL (indicated by the T) are the connections to the terminals in a cell. You will only use a TERMINAL routing region at the source and destination(s) of a route. A TERMINAL routing region, represented by a GRlocation node, has an (X,Y) coordinate identical to the cell it is on. The “H” regions are HTRACKs, horizontal routing channels between two cells in adjacent rows. It takes the (X,Y) coordinate of the cell above it. The “V” regions are VTRACKs, vertical routing channels between two cells in adjacent columns. It takes the (X,Y) coordinate of the cell to the left. Most of your routing will be through HTRACK and VTRACK regions, to get from the source to the destination(s). Note that you can route through TERMINAL locations, but only if that cell is a source or destination of the net being routed. In the next diagram we will show which routing regions can be connected together directly. However, you really don’t need this information – the class EnumerateGRneighbors is provided to automatically return all of the legal neighbors of a given routing region.

In Figure 2 we show the legal neighbors of a routing region, ignoring boundary cases. A TERMINAL can connect to the HTRACKs above and below it. An HTRACK connects to the terminals above and below it, the two HTRACKs horizontally adjacent, and the four VTRACKs diagonally adjacent. A VTRACK connects to the two
VTRACKS vertically adjacent, and the four HTRACKs diagonally adjacent. There are no routing regions beyond the edges of the chip. The exact rules for this are given in NetGlobalRouting.java.

<table>
<thead>
<tr>
<th></th>
<th>T(x,y)</th>
<th>V(x,y)</th>
<th>T(x+1,y)</th>
<th>V(x+1,y)</th>
<th>T(x+2,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H(x,y)</td>
<td>H(x+1,y)</td>
<td>H(x+2,y)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T(x,y+1)</td>
<td>T(x+1,y+1)</td>
<td>T(x+2,y+1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H(x,y+1)</td>
<td>H(x+1,y+1)</td>
<td>H(x+2,y+1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T(x,y+2)</td>
<td>T(x+1,y+2)</td>
<td>T(x+2,y+2)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. Naming convention for the global router’s routing regions. The standard cells are located at the positions designated by T’s.

Figure 2. Legal neighbors for a TERMINAL (left), HTRACK (middle) and VTRACK (right) routing regions.

**Programming Assignment**

In this programming assignment you will need to implement the function routeNet in GlobalRouter.java. You should look at the GlobalRouter.html page produced by Javadoc to understand what needs to be done, and what routines will be useful in your efforts.

As discussed above, the rules for what routing regions can be directly connected are complex – use EnumerateGRneighbors to simplify this process. This should allow you to ignore almost all of the details of the interconnect structure.
There are several diagnostic routines to help in your coding. printNGRRouting takes the routing tree you have created and prints it, helping you trace out the routing. Also, testRouting in NetGlobalRouting.java can check whether a complete route actually is a valid routing for the net. It checks connectivity, as well as making sure that all destinations are reached. You may want to call this at the end of routeNet.

This programming assignment will require understanding the public and protected functions (those that appear in the .html files from Javadoc) for NetGlobalRouting.java, EnumerateGRneighbors.java, GRlocation.java, and NGRdfs.java. You should only need to actually modify routeNet().

**Experiments**

Once your algorithm is running, you should experiment with routing after good placement. Run the simulated annealing algorithm, then call your global routing function. Also, if you are having trouble with your algorithm, remember that you can hand-place nodes via the node dialog box. Using a small benchmark (such as inverter.circ) and the ability to hand-place elements may help you develop simple test cases to help debug your router.