Final Project

CAD of Digital Circuits and Systems
ECE/CS 5740/6740

Frederico Butzke
Spring 2014

I. Introduction

This work aims to develop a channel routing software to perform a Left-Edge algorithm. This algorithm generates a track assignment for a specific channel based on *zone representation* and *vertical constraint graphs* that are generated from the pin locations.

The Figure 1 shows an example of channel routing. It has a pin set composed of {1, 2, 3, 4, 5} and each one of those has a respective horizontal track(drawn in magenta).

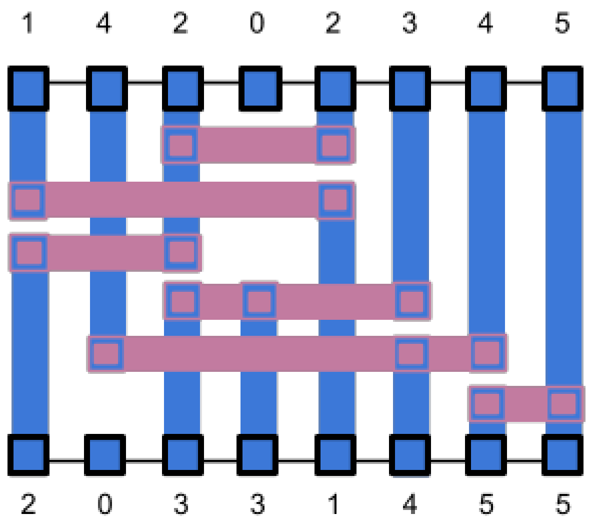

Figure 1 - Channel Routing.

As mentioned in [1] : "Channel Routing is a special case of detailed routing where connections between terminal pins are routed within a routing region that has no obstacles."

In this type of problems the pins are placed on the top and bottom of the channel and the tracks are oriented horizontally. To connect a track to its pins, vertical wires are created allowing each specified pin or netlist to connect to its track.

The vertical constraint graph is a graph *G* generated from the pin's location in the channel. It checks which pins are above others and with that information creates the vertical constraint graph, VCG. One example is shown in Figure 2. Each top pin is directed toward the bottom and then each pin is treated as a node. If there exists a path between two nodes then we do not have to specify an edge connecting both nodes. For example, $2 \rightarrow 3 \rightarrow 4$, 2 has connection with 3 and 4 but we just need to specify an edge from 2 to 3 since 3 is parent of 4 as well.

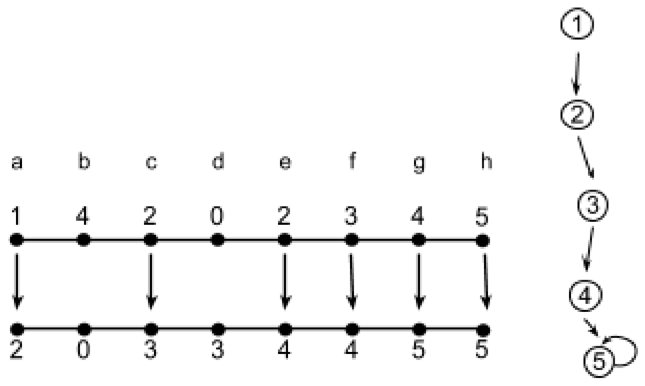

Figure 2 - Vertical Constraint Graph.

The Zone Representation allows the software to have an initial assignment of tracks and plan ahead the minimum number of tracks the problem has as well as help the tool to solve the assignment conflicts. One example of zone representation is shown in Figure 3. Here we have created a horizontal track from the initial to the last pin for each pin in the netlist. Than we found the supersets that comprised the maximum number of horizontal tracks and with that information we created the zone representations(right in the Figure 3).

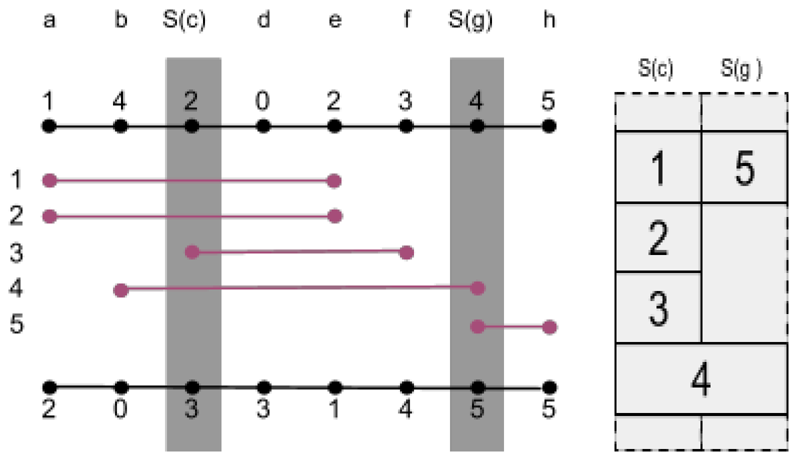

Figure 3 - Zone Representation.

The main idea of the Left-edge algorithm is that it tries to assign first the VCG's roots to the first tracks. It checks the zone representation to ensure no conflicts. Then it assigns the netlist to the specified track.

II. Algorithm

This work has developed a software tool to generate the track assignment using the Left-edge algorithm. The algorithm is shown at Algorithm 1.

Algorithm 1 - Implemented Left-edge pseudo-code.

```
1   top, bottom, netlist = input_data
2   parents, nodes = getVCG(top, bottom)
3   zones = getZoneRepresentation(top, bottom)
4   while netlist :
5           for p in parents :
6                   if noConflict(p, zones, current_net) :
7                           current_net.append(p)
8                           netlist.remove(p)
9           parents = updateVCG(parents, current_net, nodes)
10          final_zone.append(current_net)
```

At line 1 we have the data being read from the source file that contains the pins locations and order. With that we generate three informations for the algorithm. We get the top and bottom pins in a list. And the set of pins in the netlist list. The latter is required since we could have a pin just on top or just on bottom.

At line 2 we get informations with respect of the VCG. We have *parents*(the root nodes) and the *nodes*. We use the parents as a tracking variable for the top nodes of the list of nodes. The *nodes* is represented as a hash table where each node has descendents and parents depending the place they are. Leaf nodes do not have descendants and root nodes do not have parents.

At line 3 the zones are assigned. The algorithm creates a "virtual" column for each pin and then for each pin assigns its positions to the columns it is. In the end the algorithm finds the columns that are superset of others and return the zone representation.

At lines 4-10. For each parent node(line 5). If it has no conflict between the node *p* and the *current_net*(the set of current nets assigned to a specific track) in the zone representation *zones* then add the node or pin *p* to the *current_net* set. With that remove *p* from the netlists *netlist* left to be computed(line 8). With the track created and some, or all, parents assigned to the track we need to update the VCG and the parents list(line 9). To update the VCG we get the actual parents the current_net and the actual nodes and verify which parents are assigned to the current_net and then we remove them. After that we check which nodes does not have parents in the nodes list and than we set them as parents. Finally at line 10 we append the *current_net* to the *final_zone* that is the set of the netlists in each track.

III. Software

The Algorithm 1 was developed using a scripting language called Python. The *input_data* described earlier in Algorithm 1 is the only information the circuit needs to (i) create VCG and zones (ii) assign the netlists to the tracks. The data input has the format shown in Figure 4.

```
.top
0 A D E A F G 0 D I J J
.bottom
B C E C E B F H I H G I
.end
```

Figure 4 - Input Data Format.

To execute the algorithm it is required to have Python 2.7.4 or newer installed and to give the execution permission to the file ***left_edge.py***.

Once we have the permissions and the minimum required Python version we are set to execute the algorithm by using the following command: **./left_edge.py -i <input data file>**

The input file that is the default and that is inside the project folder is called *netlist.data*.

The algorithm has a verbose mode that allows us to track what is happening and the zone representations and VCGs the software is creating by using the option ***-v*** as an argument.

Once we execute the code using the previous command we get the output shown in Figure 5.

```
Solution:
Track 1  has the following netlists:  ['A', 'J']
Track 2  has the following netlists:  ['D']
Track 3  has the following netlists:  ['E', 'G']
Track 4  has the following netlists:  ['C', 'F', 'I']
Track 5  has the following netlists:  ['B', 'H']
```

Figure 5 - Output Log.

At the output log we have each track being enumerated and displaying the set of its netlist. Always the name of the pin is the same than the netlist name.

In addition an user interface will be displayed showing the channel routing mapped to a given technology in silicon.

IV. Conclusion

A good channel router is absolutely necessary nowadays. Good algorithms generate fewer tracks and optimize the places in which less wirelength are needed. Left-edge algorithm showed to be a good approach for solving this type of problems.

In addition to prove our software tool perform the routing correctly the solution shown in Figure 5 for the given pin arrangement shown in Figure 4 is the same solution the algorithm at [1] p. 177 found. Here we ended up with a channel height of five, since we have five tracks.

V. Future Work

The author of this work has not had enough time to fix the graphs which presented cycles on its nodes. To generate a channel routing with cyclic graphs we need an extra logic to split a given netlist in this cycle and assign an extra vertical graph to connect the both tracks from the same netlist.

VI. References

[1] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, VLSI Physical Design - From Graph Partitioning to Timing Closure. Springer, 2011