# Design Automation -2

Placement and Routing

Michael Shuster
M08396708
Vamsy Gedela
M12905974

## Usage

To run the program, place benchmarks in the data folder, the file extension of these benchmark files does not matter.

To run, use *./run <benchmark.*> <output.mag>*

Runtime data will be exported to the logs folder.
A magic file titled <*output.mag*> will be generated in the output folder.

The program requires these three folders (data, output, logs) to be present to run.

To recompile the program, use *./compscript*

## Placement - Force Directed Placement

For the placement algorithm, we implemented a force directed graph drawing technique. In this technique, the forces are modeled on every node such that other nodes repulse the node, and edges attract the node. The nodes are then moved based on these forces, iterating until an equilibrium state is reached in the grid.

Through this, the grid was sized based on the number of nodes so that the grid was always larger than the number of nodes. The placement grid was implemented as a map so that each cell could be accessed by a key using an encoded coordinate system where x*size + y = key. Using a map in place of a grid allows for faster searching for empty cells, and will add to stability and speed in larger datasets since vectors become slower to access than maps after around 100 elements.

During the algorithm, a map storing the nodes yet to be paced, and a multimap containing the passthrough cells are created. These are cleared after each passthrough. These structures allow for the 4 conditions of placement:

Unmoved cell,
Placing in passthrough cell,
Placing in occupied cell
Placing in empty cell

## Routing - Hadlock

Routing was done using a modified Hadlock's algorithm. In this algorithm, waves are propagated through. Each expansion starts from the lowest wave value present, and increases the wave value if the expansion is further away from the target. In our implementation, we also

increase that value if the expansion is not in the center of the channel, if the expansion is metal 2, and if the expansion is not moving in the same direction. This propagates until the target is hit, in which case the algorithm then follows the wave back choosing the lowest wave value until the source is reached. If the source is reached, the wire is then built, otherwise, the wire propagation terminates.

During this algorithm, we used a 3d vector to store the working grid of the circuit. We also used a queue to store unplaced nets, and a priority queue to store the wave propagation. Any expanded cell was placed in a stack that was then reset after each iteration, so that the whole grid did not need to be indexed to delete waves.

Wave propagation only expands to valid moves. On each expansion, the adjacent cells are checked to make sure wires are not too close to one another. This is done in wave propagation to make the recursive backtracking simpler, and quicker.

## Table for Benchmarks

| **Benchmark** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|
| Bounding Box Dimension (lambda) | 414 | 558 | 1038 | 1230 | 1518 | 1566 | 1710 | 1902 | 2190 |
| bounding box area (lambda$^2$) | 171396 | 311364 | 1077444 | 1512900 | 2304324 | 2452356 | 2924100 | 3617604 | 4796100 |
| number of feed-through cells inserted | 14 | 21 | 41 | 25 | 61 | 24 | 25 | 21 | 25 |
| total wire length (lambda) | 5769 | 13215 | 74928 | | 203937 | 242553 | 262794 | 341259 | 402819 |
| number of vias in the wiring | 36 | 100 | 628 | | 1622 | 2041 | 2143 | 2683 | 3204 |
| execution time (s) | 1.012 | 3.456 | 42.621 | | 123.027 | 159.085 | 256.212 | 241.963 | 434.052 |
| memory used (Bytes) | 4736 | 5596 | 9492 | | 16324 | 16968 | 19684 | 23484 | 29680 |
| **Routing %** | 44 | 38 | 28.75 | **#** | 22.75 | 16.833 | 13.6 | 18.8 | 13.7 |

**#** - Segmentation fault

## Retrospective

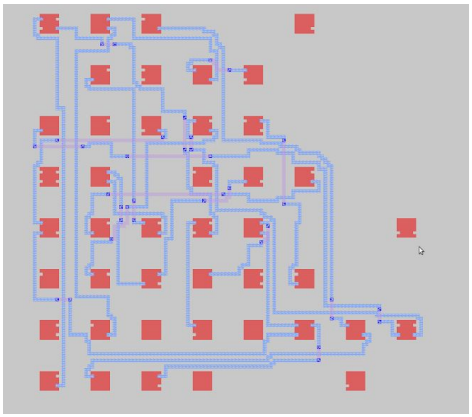
On some of the benchmarks, the program would throw a segmentation fault. This is due to the backtracking stage not finding a valid path to the source. Given more time, we would be able to implement a stack into the backtracking stage that would save possible branches to correct this issue.

The net placement percentage is very low in our algorithm. In our current implementation, waves are weighted higher if they are outside of the center of the channel, however, this is only affecting a fixed interval. Due to this, wires will always try to get within the center of the channel, but if the channel is full, they will not gravitate towards the center, but rather will choose the

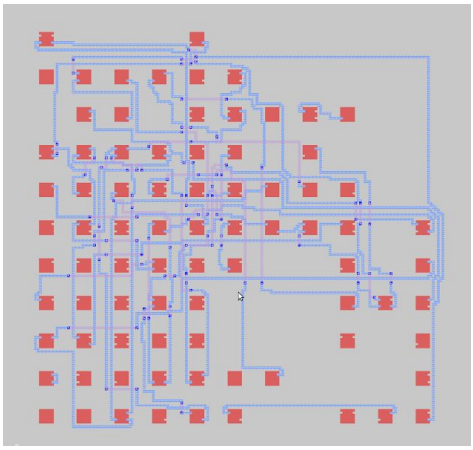
default path of hugging the nodes. This could be fixed by implementing a logarithmic scaling value to the incrementing waves based on how far from the center of the channel they are.

The routing algorithm is also very poor on memory because it stores an entire working grid for the routing process. This could be improved by using a map based system like in the placement algorithm. With this system, empty cells could be omitted from the map, and only cells with data would be stored in the map. This would allow for only rendering the important parts of the grid.

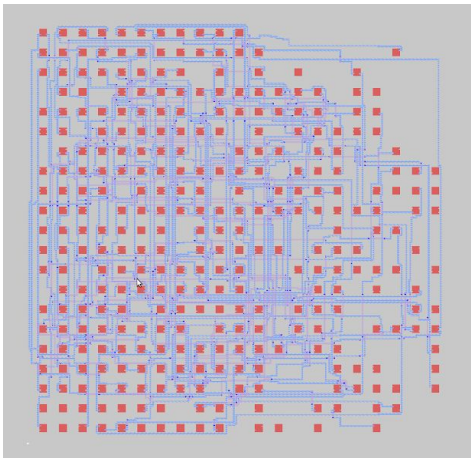
Routing could also be improved by implementing a channel routing based algorithm. With this, we could also get wire estimates prior to routing, and base placement around creating passthrough cells only where we estimate to be needed. This would add to the density of the circuit.



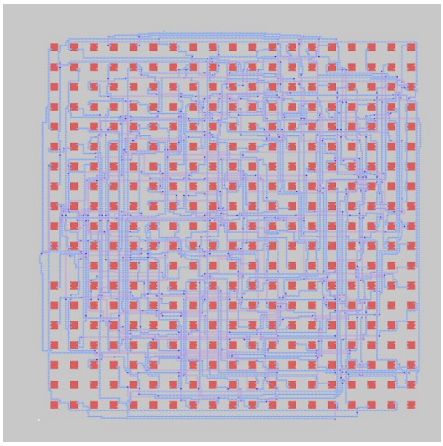
1.mag



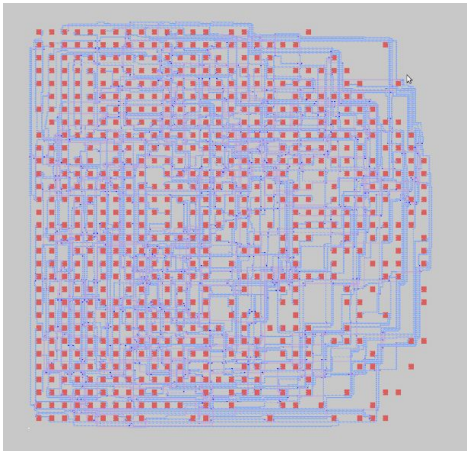
2.mag



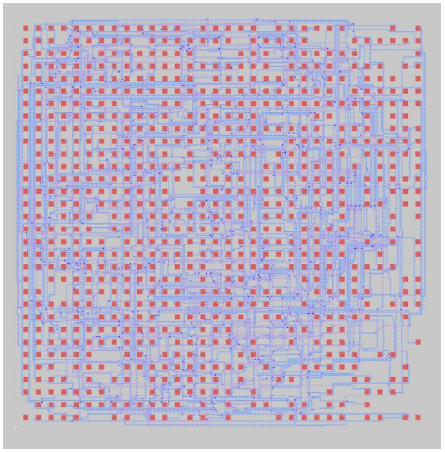
3.mag



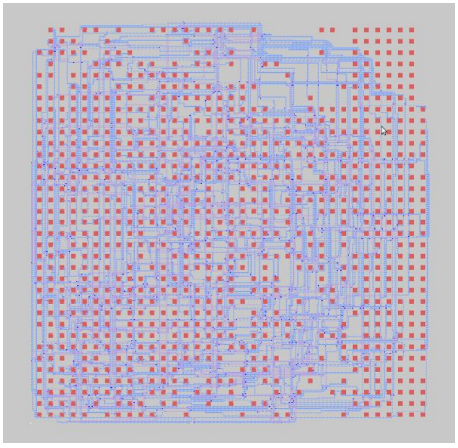
4.mag



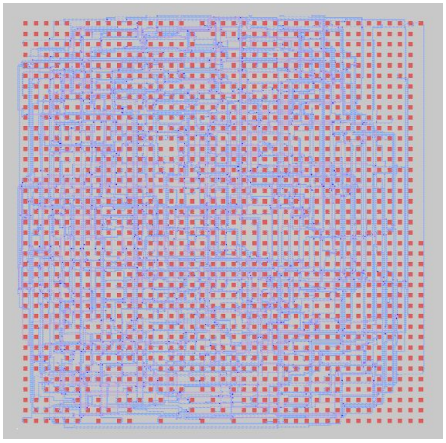
5.mag



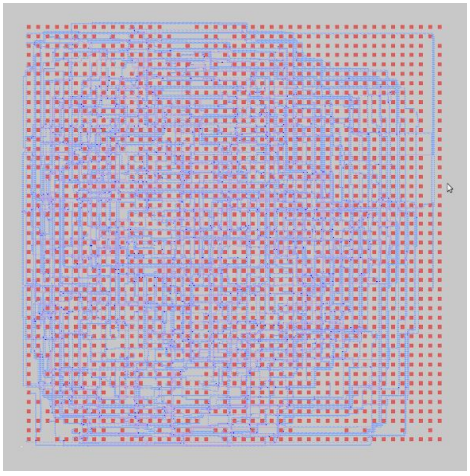
6.mag



7.mag



8.mag



9.mag