VLSI DESIGN AUTOMATION

HOMEWORK – 2

Placement and Routing

SAI CHAITANYA NANDIPATI

M08905912

nandipsa@mail.uc.edu

ASRITH REDDY SINGA REDDY

M08893106

singaray@mail.uc.edu

Placement

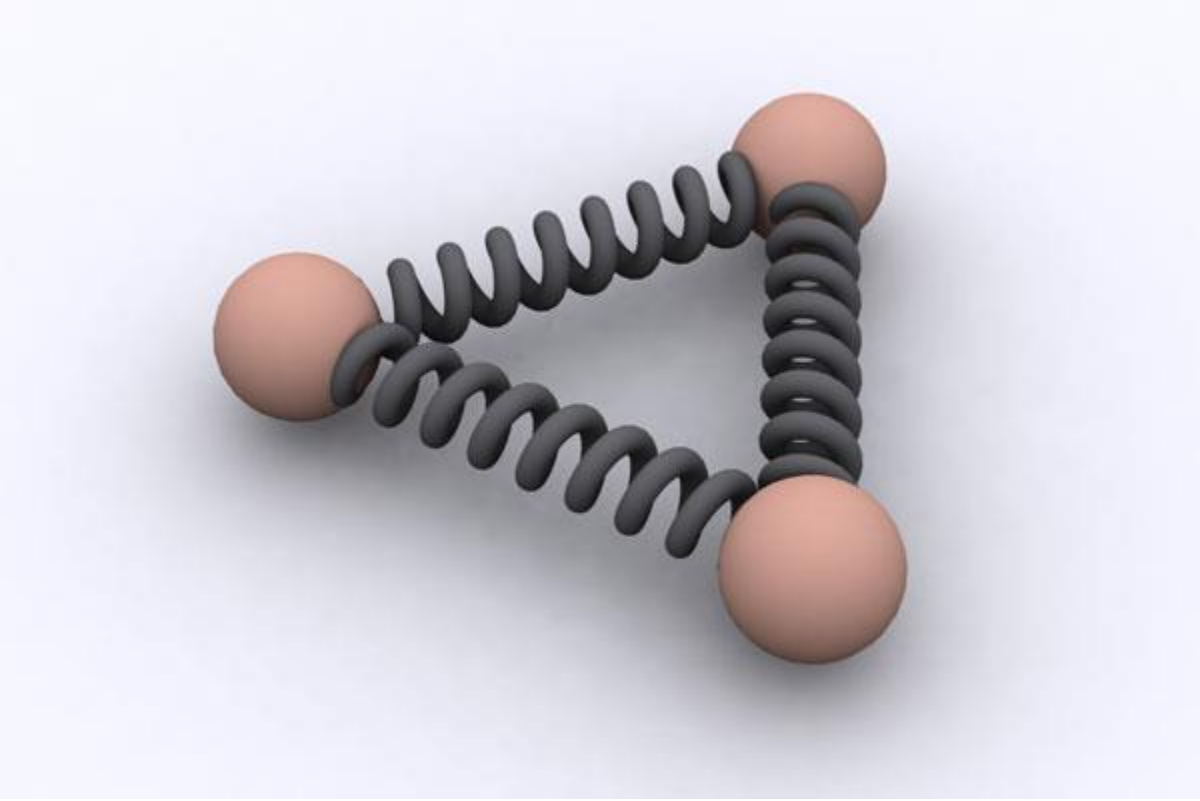
Method Used: **Force-directed placement**

Rationale for using this method: Other methods available are simulated annealing, min-cut method and methods using artificial neural networks. Simulated annealing is by far the most researched method proven to be giving good results but takes time and large amounts of memory to execute. Force-Directed is simple and the results can be improved by increasing the iterations. The amount of memory and time taken are significantly less when compared to Simulated Annealing. Min-cut method is proved to be inefficient when the cells are densely connected.

Basic Idea: The cells to be placed in the layout are considered to be analogous to bodies with mass (In this case of same mass).

The nets connecting the cells are considered analogous to springs connecting the bodies.

In such a spring mass system, the bodies get displaced and move in to a final position such that the total sum of the equivalent forces on the bodies is a minimum or zero.



Algorithm:

```
Compute the total connectivity of each cell;
Order the cells in decreasing order of the connectivity and store them in a list;
While(iteration_count< iteration_limit)
        Seed= next module from the list;
        Declare the position of the seed vacant;
        While ( end_ripple=false)
                Compute the target point of the seed and round of to the nearest integer;
                Case target point is
                        VACANT:
                                Move seed to the target point and lock;
                                End ripple <- true
                                Abort_count<-0
                        SAME AS PRESENT LOCATION:
                                End_ripple=true;
                                Abort_count=0;
                        LOCKED:
                                Move cell to the nearest vacant position;
                                End_ripple=true;
                                Abort_count=abort_count+1;
                                If(abort_count>abort_limit) then
                                        Unlock all the cell positions;
                                        Iteration_count+=1;
                                Endif;
                        OCCUPIED: (AND NOT LOCKED)
                                Select cell at the target point for the next move;
                                Move seed to the target point and lock
                                End_ripple=false;
                                Abort_count=0;
                        End case;
                End While;
        End While;
End;
```

Algorithm Tuning:

The force Directed Algorithm can be used is various ways.

Ripple moves in which a cell in a target location of the other cell is selected as the next seed in the next move.

Chain move is a method in which the cell in the target location is moved to the adjacent location and any cell if present is that adjacent location is moved further to the adjacent location thereby initiating a chain move which ends when a free location is found.

1) Abort Count: The abort_count variable is set according to the number of cells in the benchmark. It is taken as the 30% of the number of cells in the benchmark.

2) Iteration Limit: The iteration limit is at most set to 10 as increasing it anymore doesn't yield any significant improvement

3) Size of the graph: The size of the graph plane is selected as the nearest perfect square to the number of cells in the bench mark.

4) Square plane Selecting the graph plane size as a perfect square allows us to make the placement as close to a square as possible.

5) Ripple move: The ripple moves used selects the cells which are out of order of the total connectivity most of the times. To go back to the actual seed order after detour, the seed order is again set to initial seed and the placement is done again.

6) Initial placement: The initial placement is done randomly using the srand random number generator with the seed taken as the system time

Data Structures Used:

Vectors and structures are used for the implementation of the data structures to allow the flexibility of adding and removing elements.

Data Structure for Cells:

```
struct cellconnections{
   int cell_number;
   int X,Y;
   int terminal_links[4][2]={{-1,-1},{-1,-1},{-1,-1},{-1,-1}};
   int connectivity=0;
};
Vector<cellconnections> cell;
Vector<cellconnections>sorted_cells;
```

Each cell is given a cell number, its position on the placement place with coordinates X and Y. Terminal links are given such that the first element in a row is the destination cell number and the second element in each row is the terminal of the destination cell.

For example: A cell 45 connected to the cells and terminals respectively in the following way (40,1)(35,2)(22,3)(11,4) has the structure values as follows.

Example:
```
struct cellconnections{
   int cell_number= 45
   int X,Y (some position on the plane)
   int terminal_links[4][2]={{40,1},{35,2},{22,3},{11,4}};
   int connectivity=4;(as it is connected to four other cells or has four nets connected)
};
```

Data Structure for the nets:

```
struct netstruct{
   int netnum;
   int cell1, cell2;
   int terminalnum1, terminalnum2;
};
vector<netstruct>net;
```

The data structures for the nets is a vector of type struct. Each structure has the elements netnumber, and the cells and terminals it connects.

For example: A net number 2 from the bench mark connecting the terminal 1 from cell 10 to terminal 3 to cell 57 will have the structure elements as follows.

Example:

```
struct netstruct{
   int netnum= 2
   int cell1=10;
   int cell2=57;
   int terminalnum1=1;
   int  terminalnum2=3;
};
vector<netstruct>net;
```

Data Structure for the placement plane:

The Data structure for the placement plane is also a vector of struct as follows.

```
struct graphplane{
   int X,Y;
   int cell_number=-1;
   bool occupied=false;
   bool locked=false;
   int sortedcellindex=-1;
   int vacantcellindex=-1;
};
```

Each struct element of graph plane contains the graph coordinates and cell number allotted to it which his initially set to -1 which means no cell is allotted to it yet.

The flags occupied and locked are used to know whether the cell is located in the particular location and whether it is locked or not.
The sorted cell index in the structure is used to store the index of the particular cell in the list of the sorted cells according to the connectivity.

The vacant cell index is used to store the index of the cell if it is vacant and its position in the vacant cell list which stores the locations of the vacant cells in the graph plane.

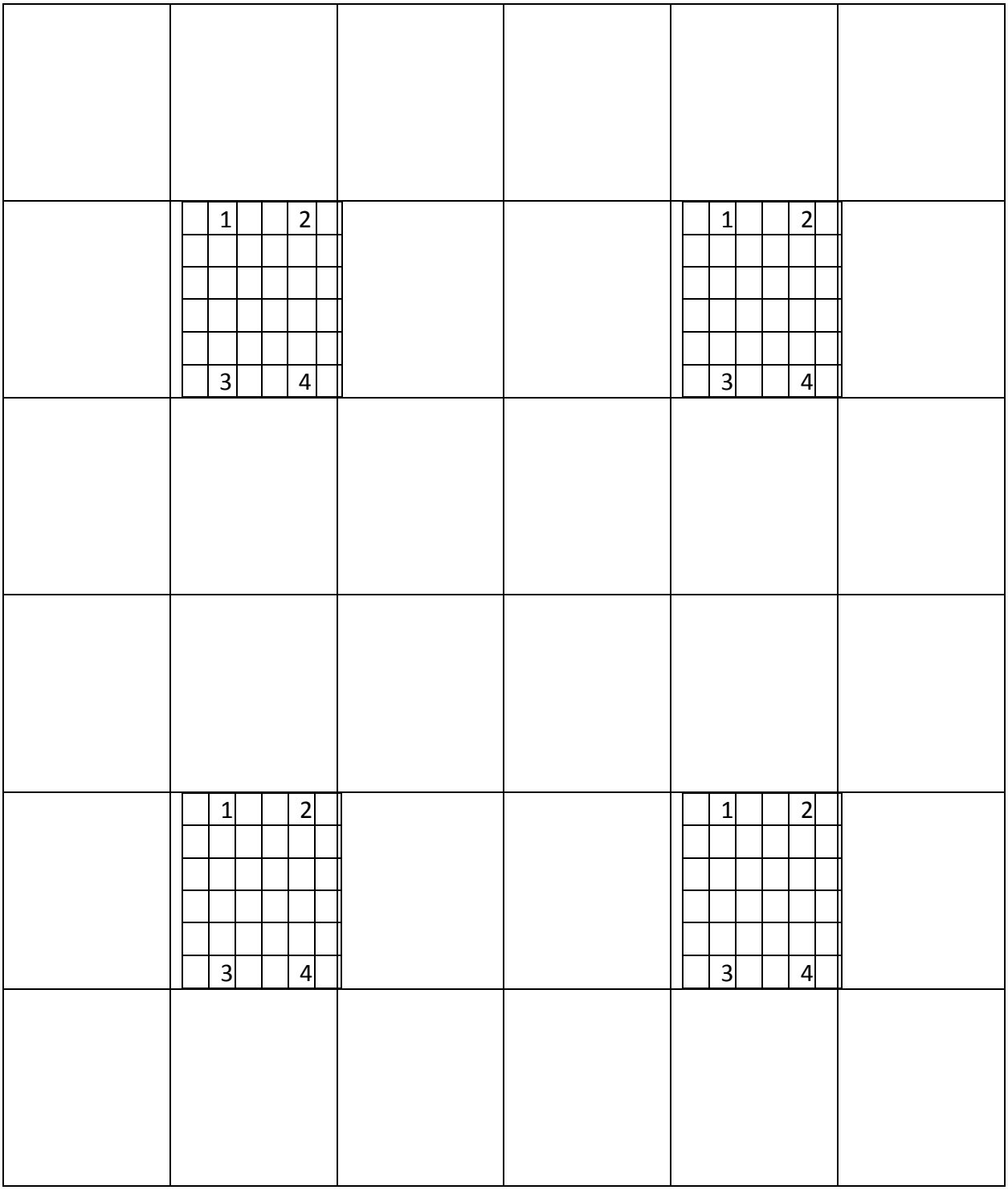
Implementation and linking with the routing

The placement is done using a cell size of 1x1 during the initial step and in the second stage the cells are expanded to their original size along with the locations of the terminal points. The cells after expanding are given a gap of 12 grids between them and also the border.

PLACEMENT BEFORE EXPANDING

CELL 1	CELL3
CELL 4	CELL2

PLACEMENT AFTER EXPANDING



ROUTING

Algorithm used:

Lee algorithm:

In the VlSI design process routing follows placement. The objective of the routing is to find and connect a path (wire) between two pins of the same net. There are two broad classes of routing algorithms. Maze routing algorithm and channel routing algorithm.

Lee routing falls under the category of maze routing algorithm.

Objective: Find suitable path on the available layout space, on which wires are run to connect the desired set of the pins.

Minimize the given objective function, subjective to the given constraints. Types of constraints include minimum spacing between wires or cells, minimize the wire length, timing constraints etc.

Lee algorithm connects two pins at a time.

Characteristics of Lee algorithm:

If a path exists between source and target terminal then it is definitely found.

It always finds the shortest path.

It uses breadth first search.

The time complexity $O(n^2)$ space complexity N*N

It mainly consists of three phases:

Wave propagation Phase:

During step I, non-blocking grid cells at Manhattan distance of I from grid cell S are labeled with i.

Labelling process continues until the target grid cell T is marked in step L. L is the length of the shortest path.

The process terminates if the target is reached or no new cell is labelled.

Retrace:

Systematically backtrack from the target towards the source.

If T reached during step I, then at least one grid cell adjacent to it will be labelled i-1, and so on.

By tracing the numbered cells in the descending order we can reach S following the shortest path.

In backtracking process most of the times there is a choice of the cells that have to be made. To minimize the number of bends do not change the direction of retrace unless one has to do.

Label clearance:

All labeled cells except those corresponding to the path just found are cleared.

This process is similar to the wave propagation step and complexity is O (L^2).

L is the length of the path.

Memory requirement:

Each cell need to store a number between 1 and N^2, N*N is the number of grids.

One bit combination to denote empty cell

One bit combination to denote obstacles.

$\text{Log}_2(N^2+2)$ bits per cells.

Describing the performance:

At first I have used metal 1 to connect all the horizontal wires and metal 2 to connect all the vertical wires.

Problems encountered: Number of vias is high.

Previously routed nets blocking the terminals of the cells.

Unable to route some nets.

Solution to the problems:

Number of vias can be reduced by reducing the number of the turns. To solve these problems ordering is given to the nets.

Net ordering:

Ordering the nets in the ascending order of number of pins with in their bounding boxes.

Ordering the nets in the ascending order of their length.

Observation: Ordering nets based on the length is producing better results.

To avoid cell terminal blockage problem all the cell terminals are initially blocked and they are cleared before they are routed.

Observation: Implementing with these constraints resulted in increase in the number of nets routed.

Alternate implementation:

First route all the nets using metal 1 and then route all remaining using metal 2.

Problems: If a cell is surrounded by both metal 1 and metal 2 then it cannot be further routed.

Solutions: Problem can be solved by switching between metal 1 and metal 2 alternatively whenever routing in one layer is not possible.

Further improvements: If I had more time I would have implemented switching between metal 1 and metal 2 and memory optimization by reducing the number of variables and improved lee numbering. Execution speed can be improved by using numbering from both source and target terminals and selecting the source which is farthest from the center. Quality of the output can be improved by reducing the number of vias and reducing wire length. This can be achieved by further optimizing the placement.

Data structures used in the code:

To store the net information from given benchmark: 2D vector of the type struct that can store source and target cell, source and target terminal and net number.

To represent the layout matrix : 2D vector in which each grid can store the coordinates, lee number, net number, cell number, horizontal and vertical blockage information.

Cell terminal information: This vector stores the cell terminal coordinates in the layout.

To represent the cell terminal coordinates:

Data structure used: The entire routing surface is represented by 2D vector of grid cells.

Grid cell is of the type struct.

Structure built with the variables requires to store horizontal and vertical constraints net number, via and cell blockage location of the cells etc.

Via variable is used to store the via (connection between metal 1 and metal 2) cell blockage, terminal blockage, cell location and cell blockage.

Vertical blockage variable is used to store the blockage information for the metal 2 and horizontal blockage variable for storing the blockage information for the metal 1.

LeeNumb is used to store the lee number.

Wave propagation phase:

Labelling process propagates from the source to the destination.

Deque from the std library is used to store the information regarding the grids to be labelled. Source grid is pushed into the Deque and Lee number is updated as 0. All the surrounding grids with respect to present location are checked and if there is no blockage and grid is not labelled yet then lee number is incremented by 1 in that grid and the grids which are newly labelled are pushed into the deque and grid whose all surroundings are verified is popped out from the

deque. This process is iteratively repeated till the target is reached. Deque is cleared once the target is reached.

Time Complexity: $O(L^2)$ L-Length of the path

Back trace:

Implementation in the code:

Backtrace process propagates from the target to source.

First target grid is pushed into the deque. All the surroundings of the target cell are verified and if the lee number is less than the target cell then it is pushed into the deque and blockage and net number is updated in the corresponding variables, and target grid is popped out from the deque. This process iteratively continues until the target is reached. To minimize the number of bends direction of retrace is not changed unless no grid is found with lee number less than 1 with respect to the present grid. This achieved using 4 while loops for 4 directions. Deque is cleared once the source is reached.

Time Complexity: O(L) L-Length of the wire from source to the destination.

Clearing phase:

Implementation in the code:

This process is similar to the wave propagation phase. Source is pushed into the deque and all the surroundings are verified and if there is no blockage lee number is cleared. This process is iteratively repeated until the target is reached.

Complexity: $O(L^2)$

L-length of the wire from the source to the target

Memory requirement for the code that I have implemented is more than the $\text{Log}_2(N^2+2)$ as I have used different variable to store the blockage information, and netnumber via, cell and terminal blockage information.

Even though code is working fine for the given benches there might be memory constraints for the bench marks with high number of nets and cells.

This can be further improved by:

Instead of using sequence 1,2,3,4,5… for numbering the cells 1,2,3,1,2,3…. Is used. Memory requirement 3 bits per cell.

Use the sequence 0,0,1,1,0,0,1,1…. can be used. Memory requirement 2 bits per cell.

Note: any numbering scheme for which labels of predecessors and successors are different can be used.

Time complexity: Time complexity is similar to the basic algorithm. $O(L^2)+O(L)+O(L^2)$.

This can be further improved by using following techniques:

Start point selection: Choose the starting point as the one that is farthest from the center of the grid.

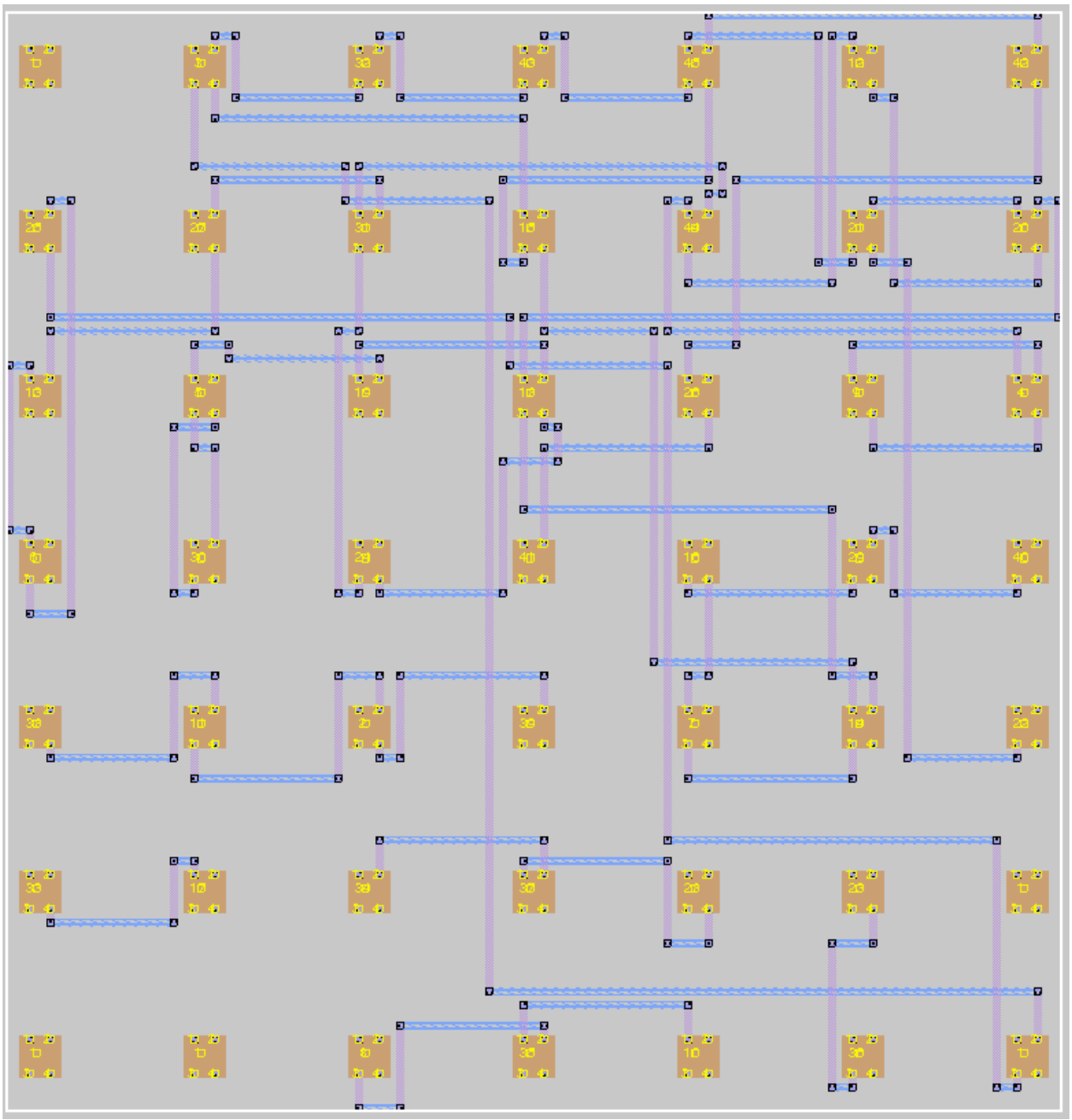
Double fan out: Propagate the waves from both the source and the target cells labelling continues until wave fronts touch.

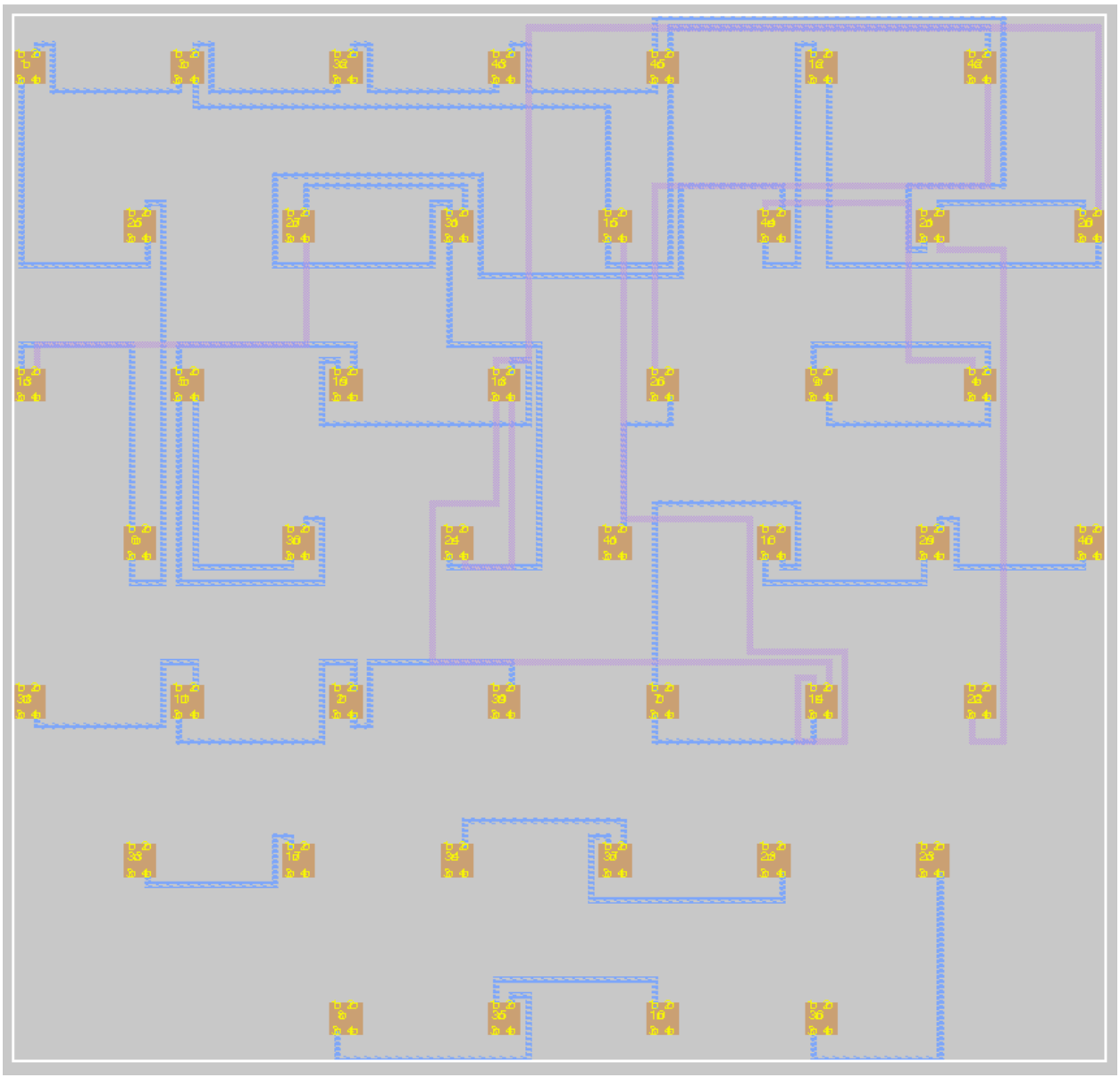
Framing: An artificial boundary is considered outside the terminal pairs to be connected. 10-20 % larger than the smallest bounding box.

Benchmark	Bounding Box dimensions	Box area	Manhattan Distance	Total wire length	Number of vias	Execution Time	Memory
B1	152x160	24320	114	2715	158	0.72s	3180kb
B2	226x234	52884	386	5776	316	2.07s	4764kb
B3	322x330	106260	990	11547	614	4.03s	7668kb
B4	442x450	198900	2055	26863	1279	13.44s	12684kb
B5	634x640	405760	4627	55317	2423	30.99s	23440kb
B6	754x764	576056	6029	79247	3377	51.02s	31848kb
B7	542x546		9881	75814	2404	3:51s	17872
B8	704x712	501248	3667	51575	2482	20.67s	28312kb
B9	682x688	469216	1275	19320	1232	4.70s	26412kb
B10	586x592	346912	530	9045	594	2.77s	20340kb

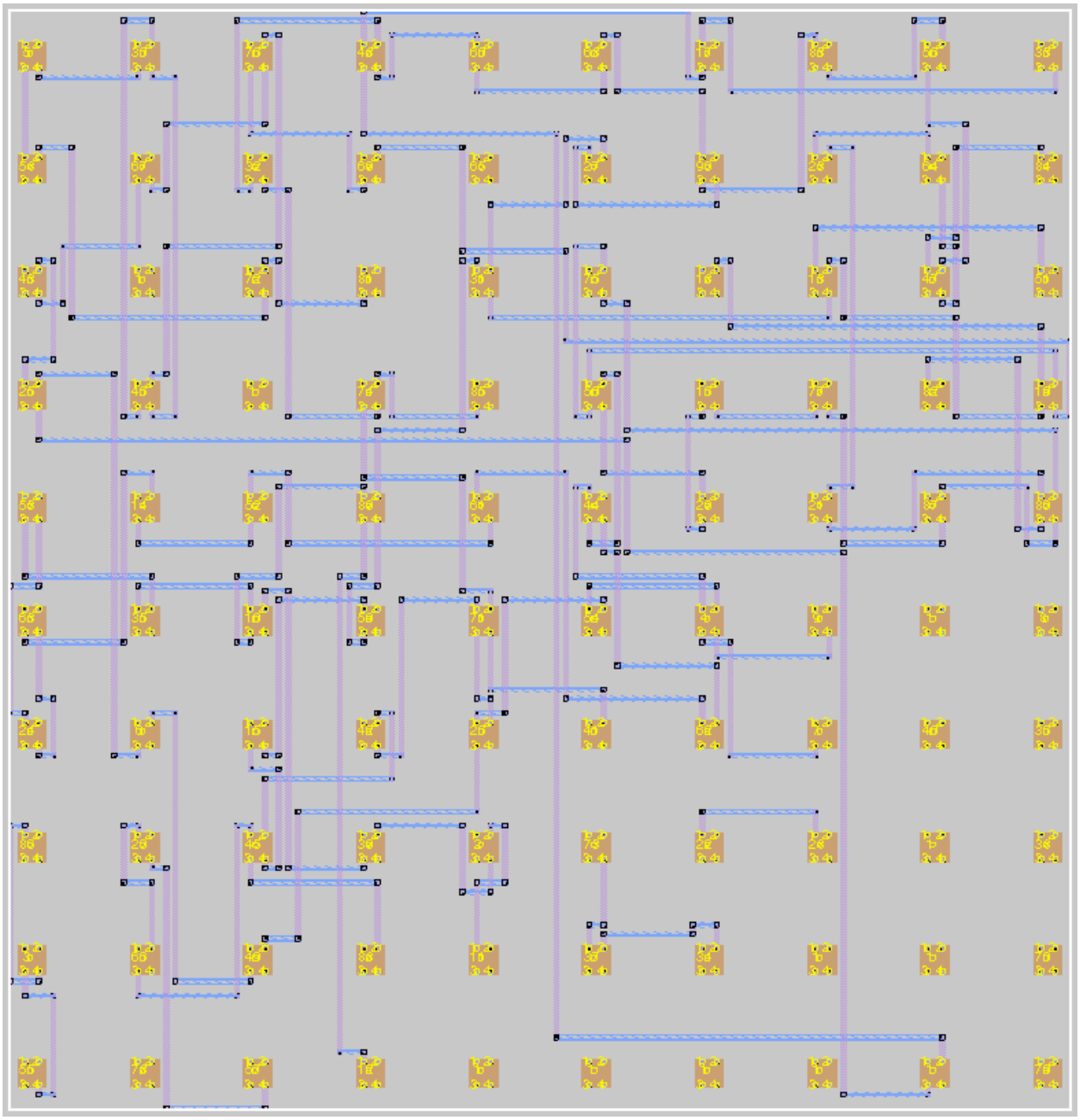
Magic Layout Diagrams

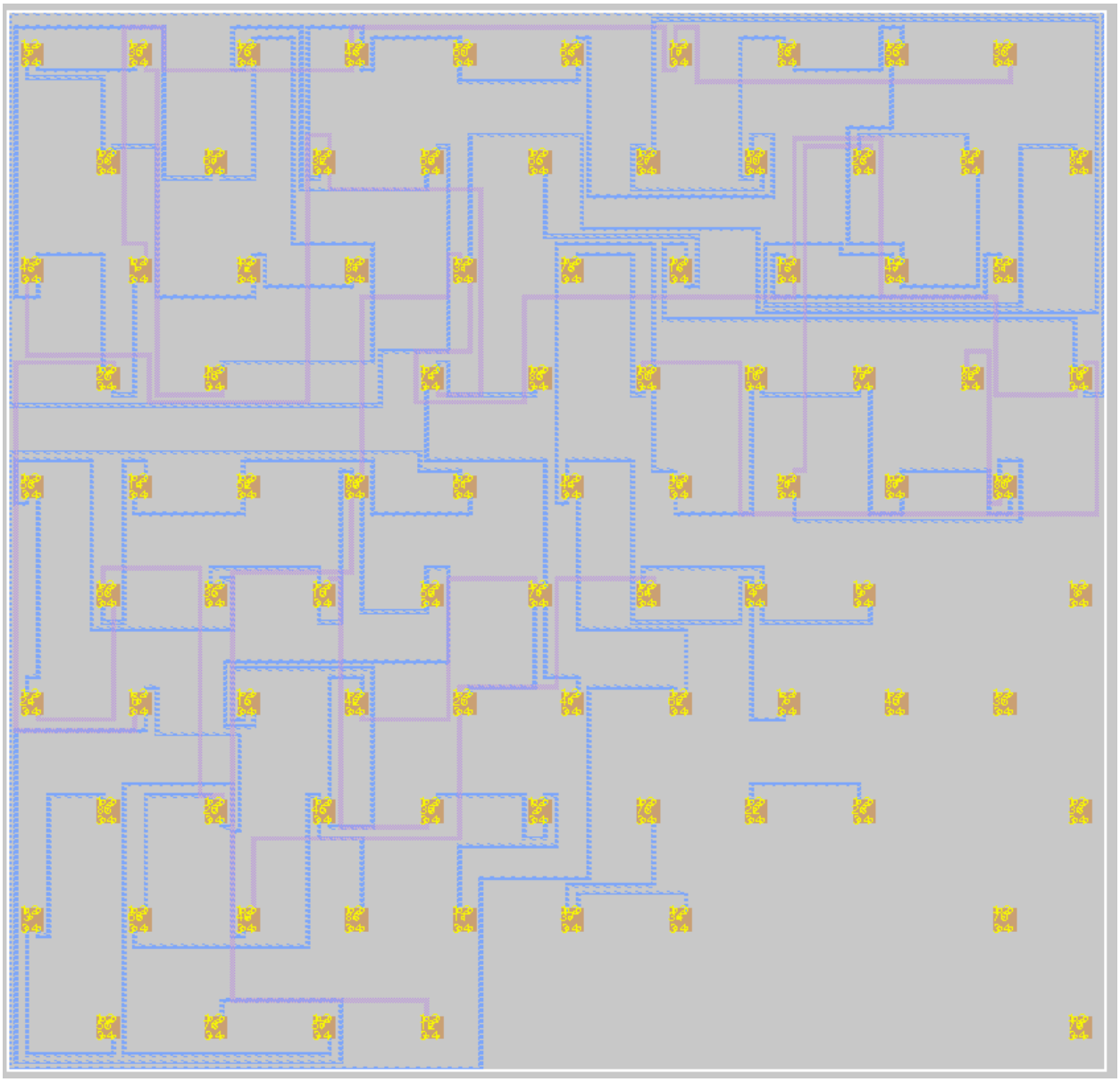
Bench Mark 1



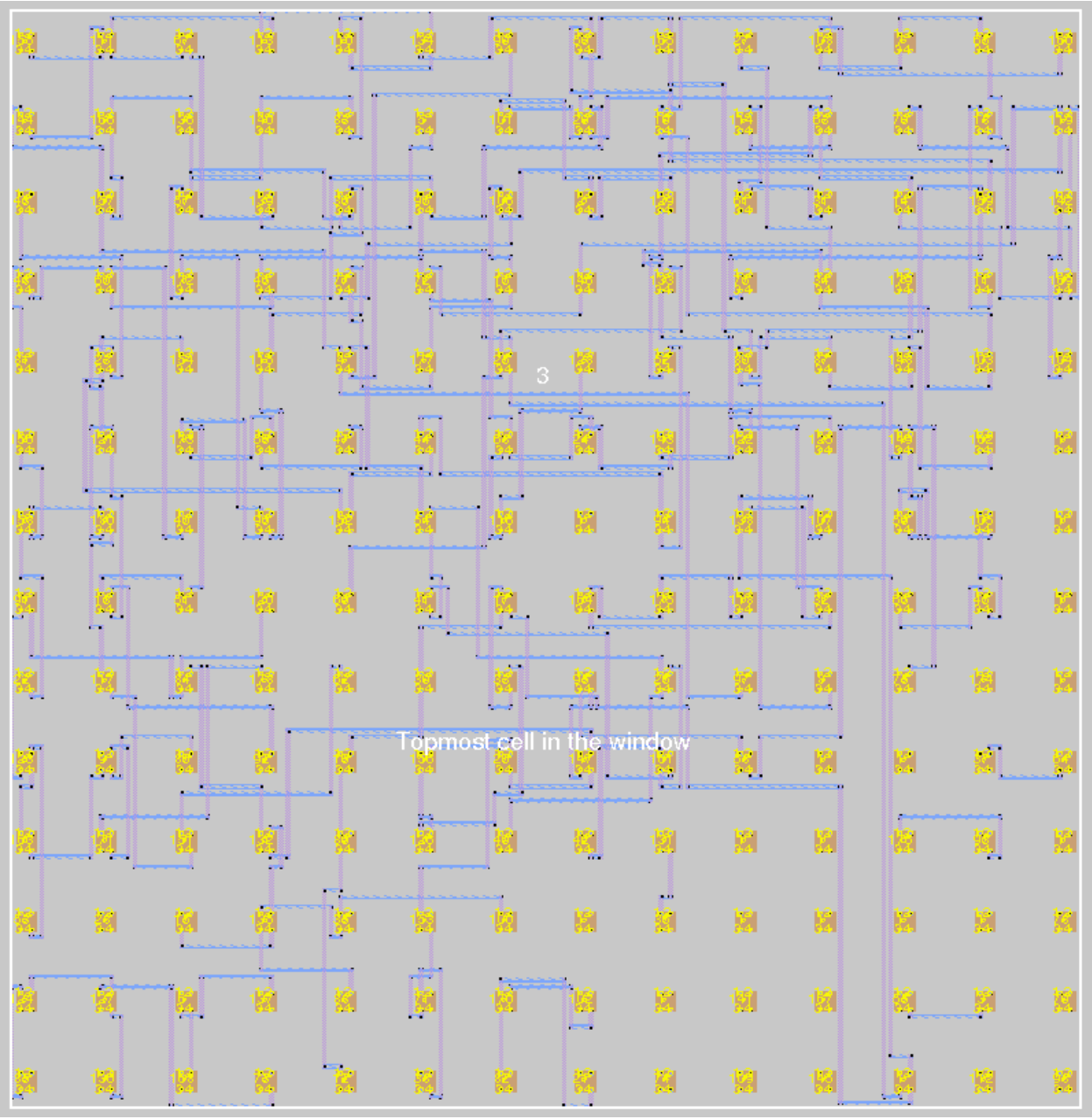


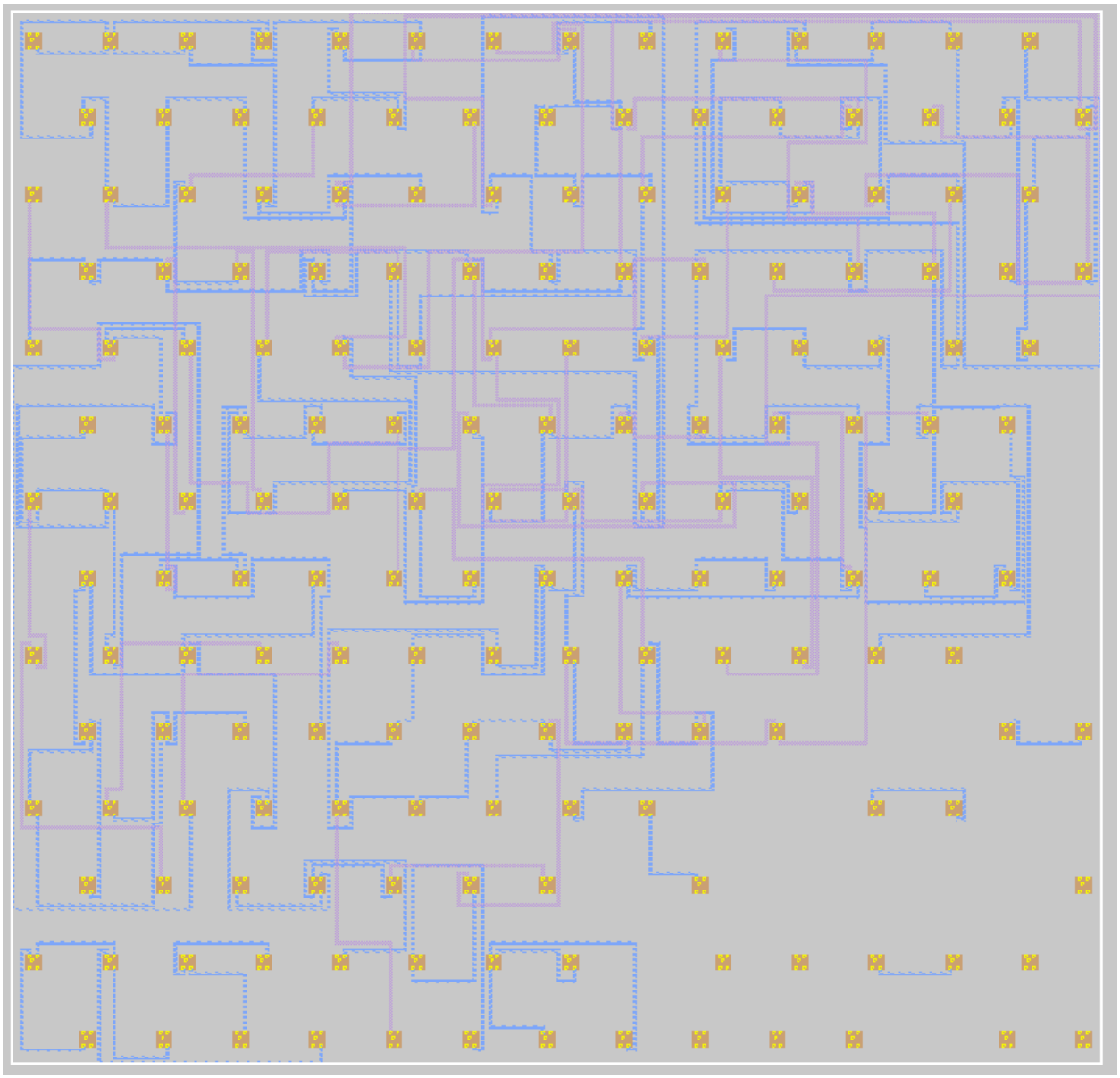
BENCHMARK 2



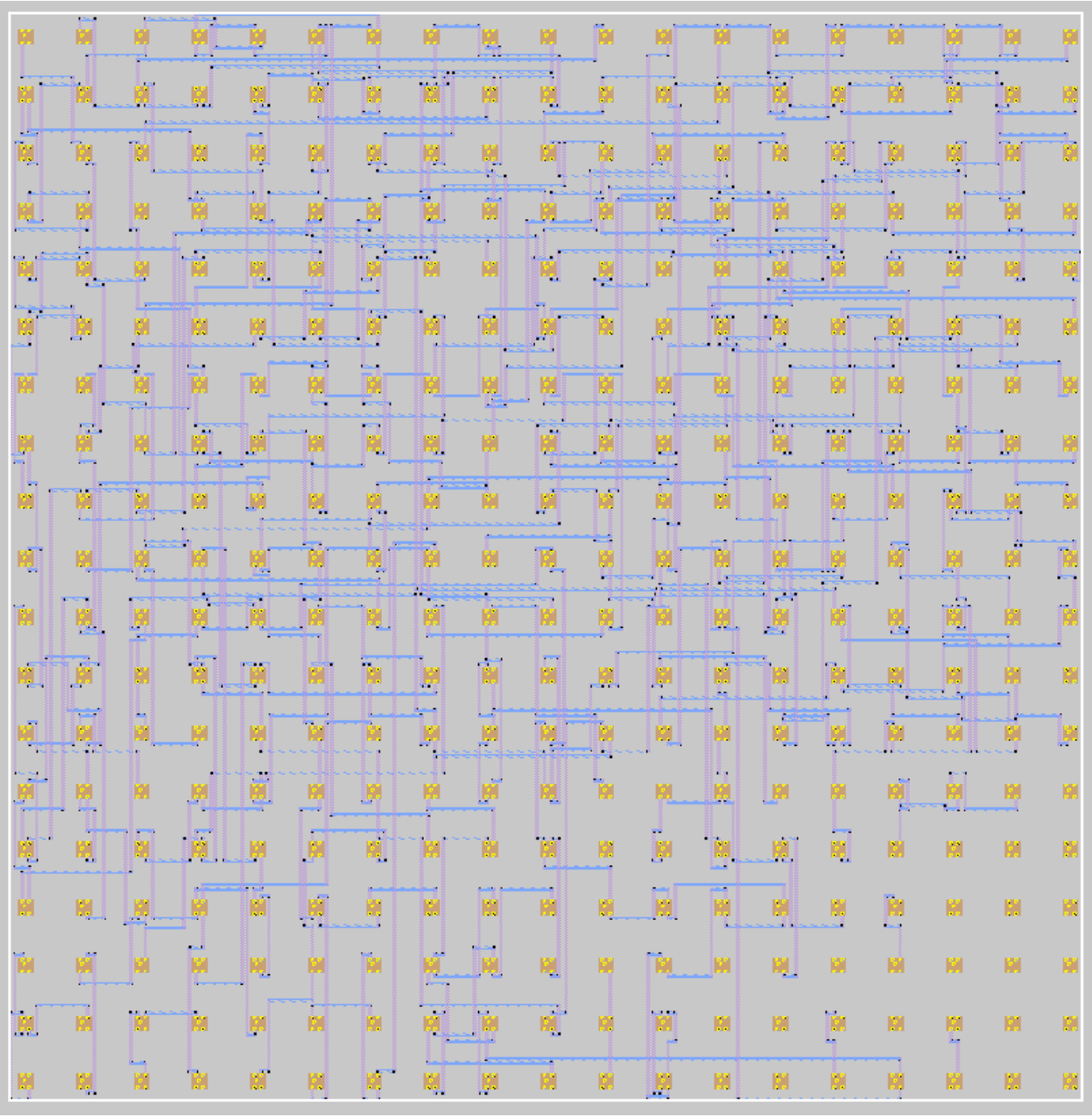


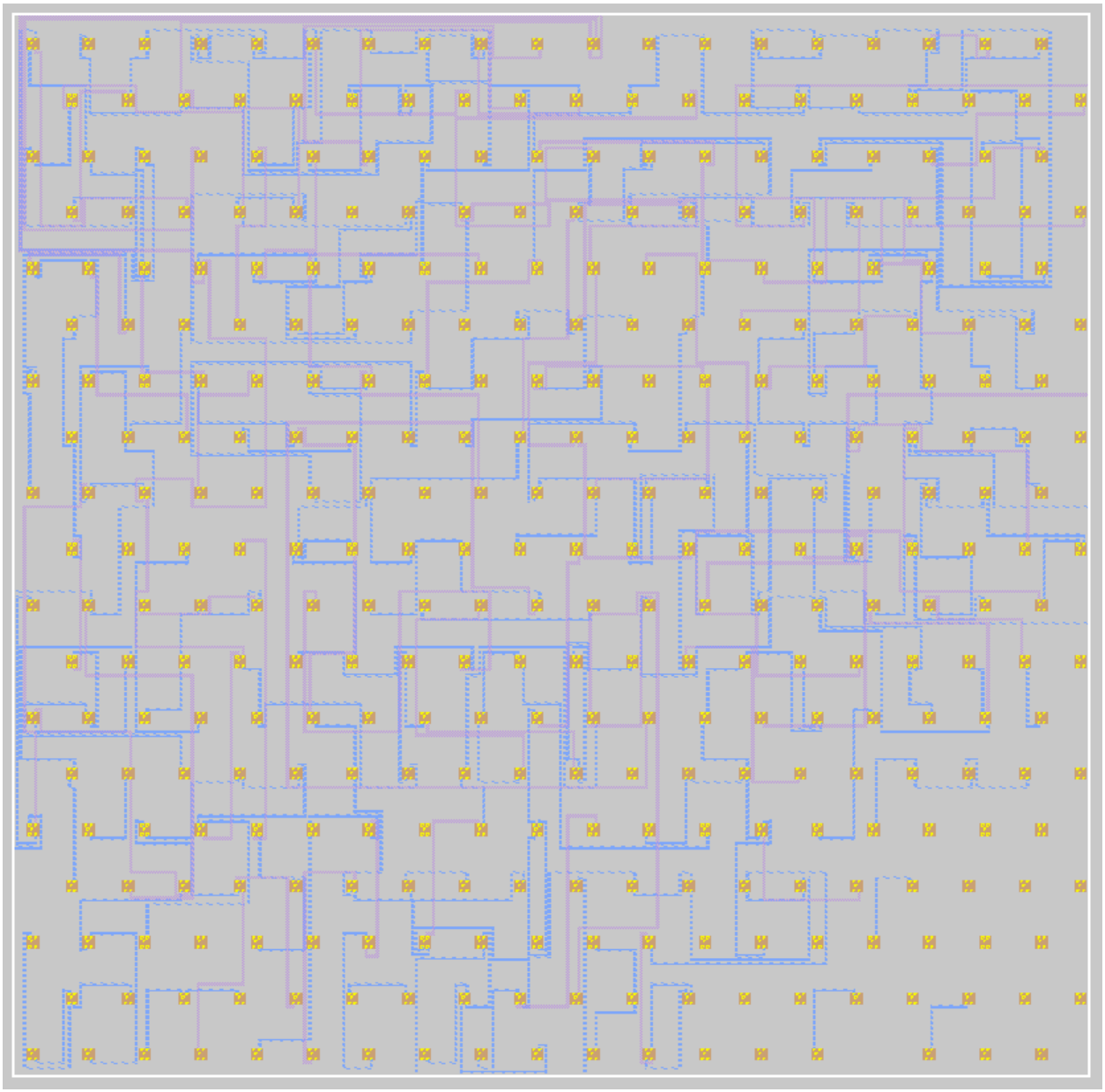
BENCHMARK 3



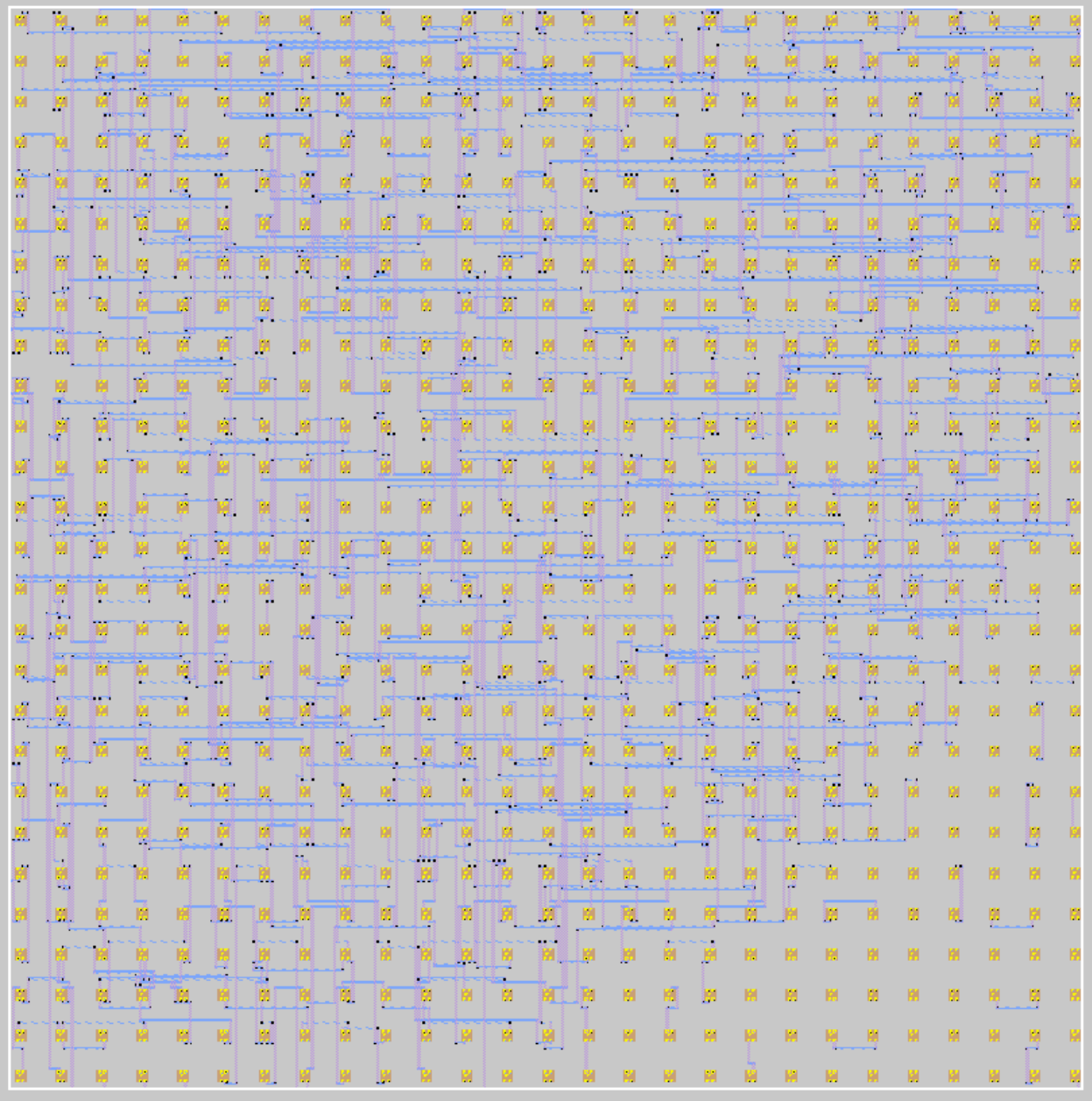


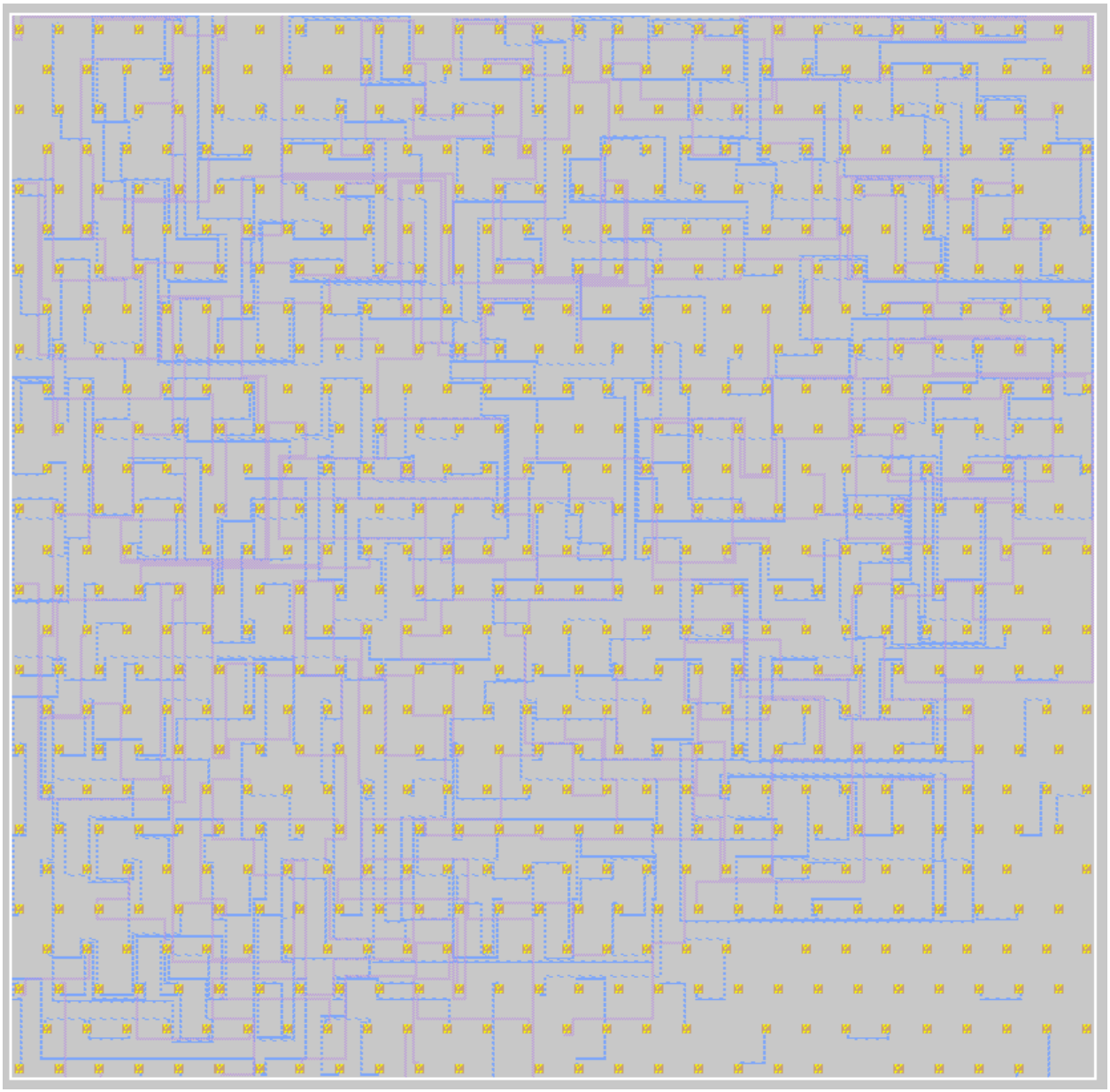
BENCHMARK 4



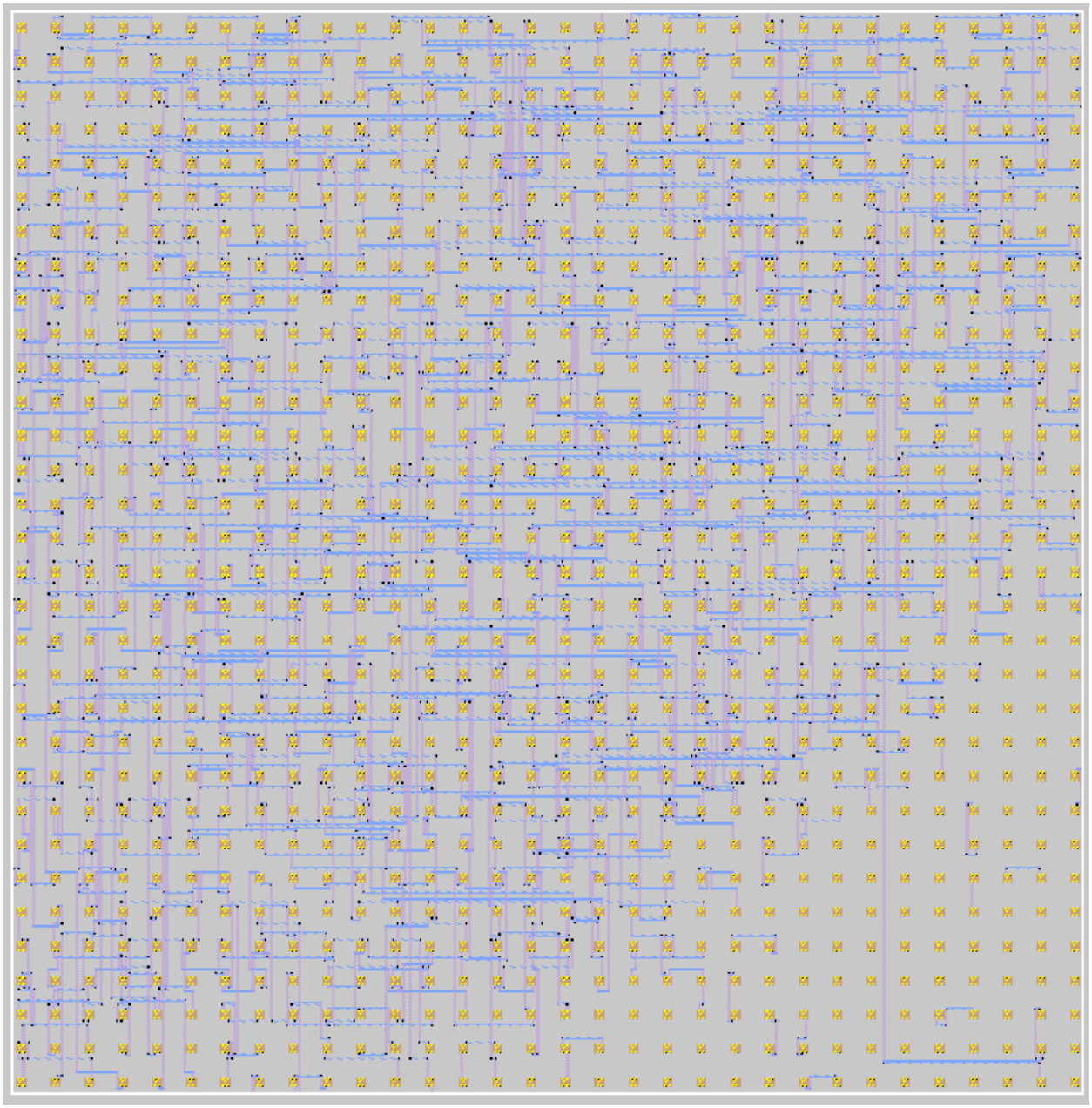


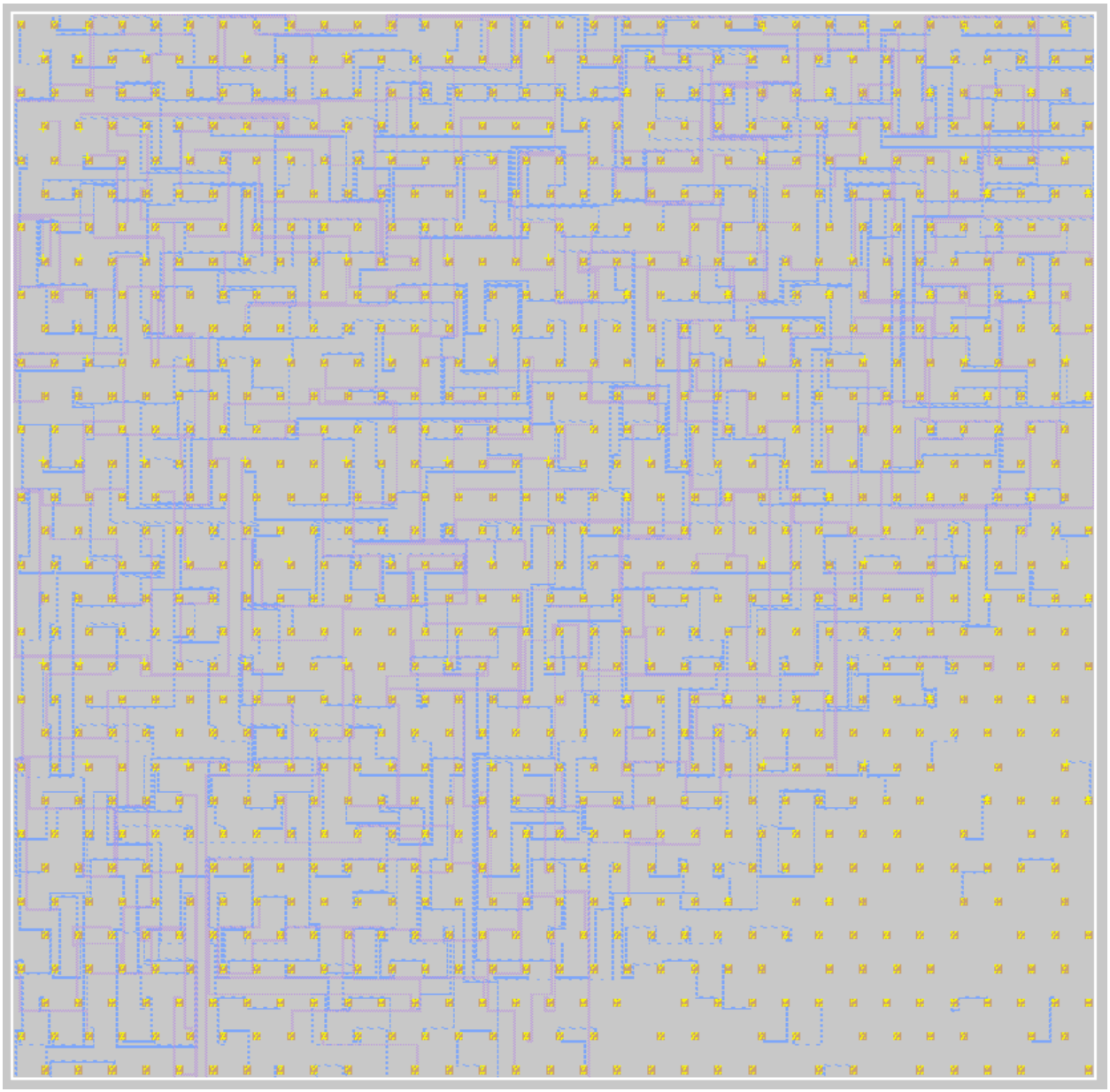
BENCHMARK 5



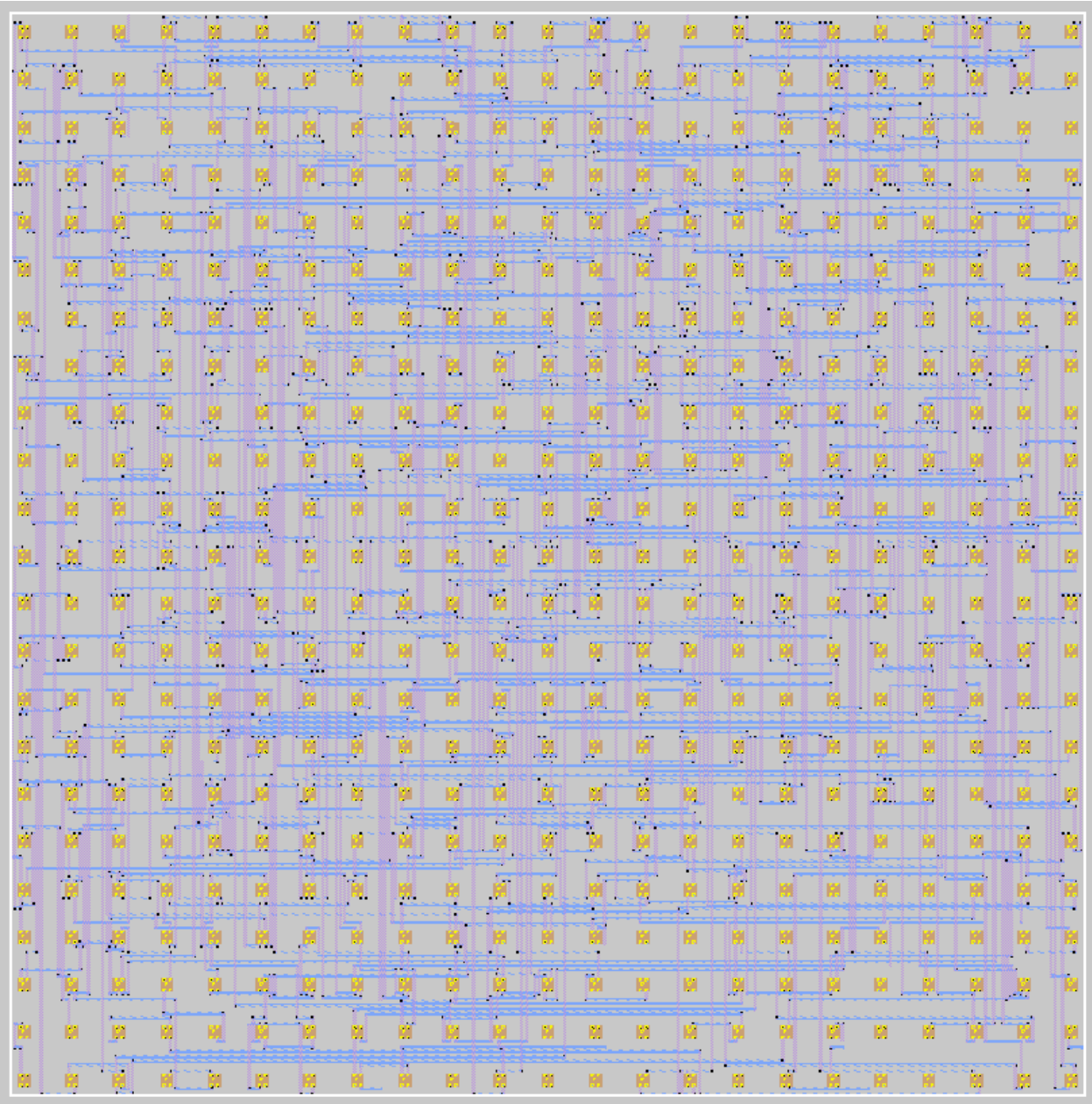


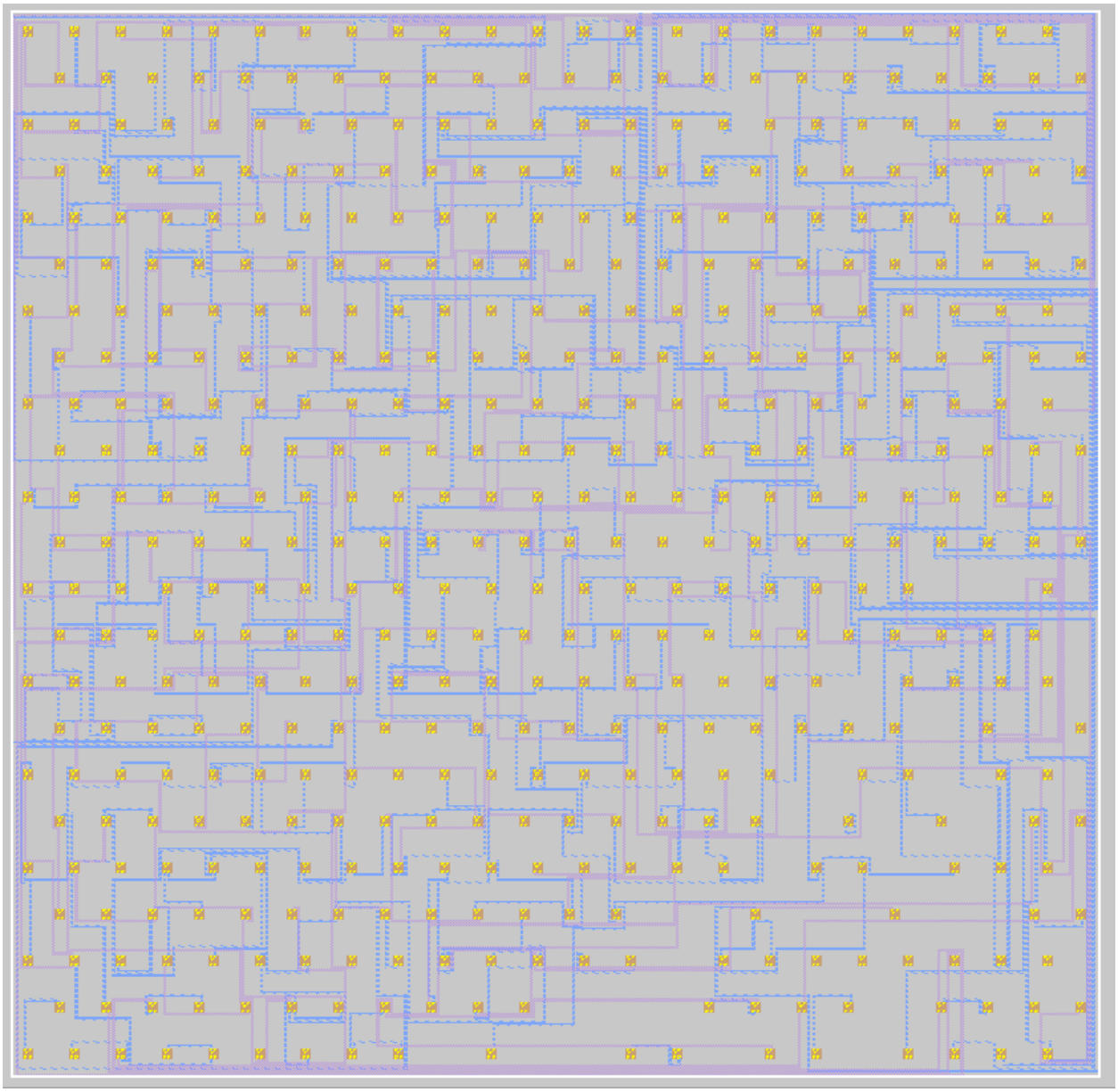
BENCHMARK 6



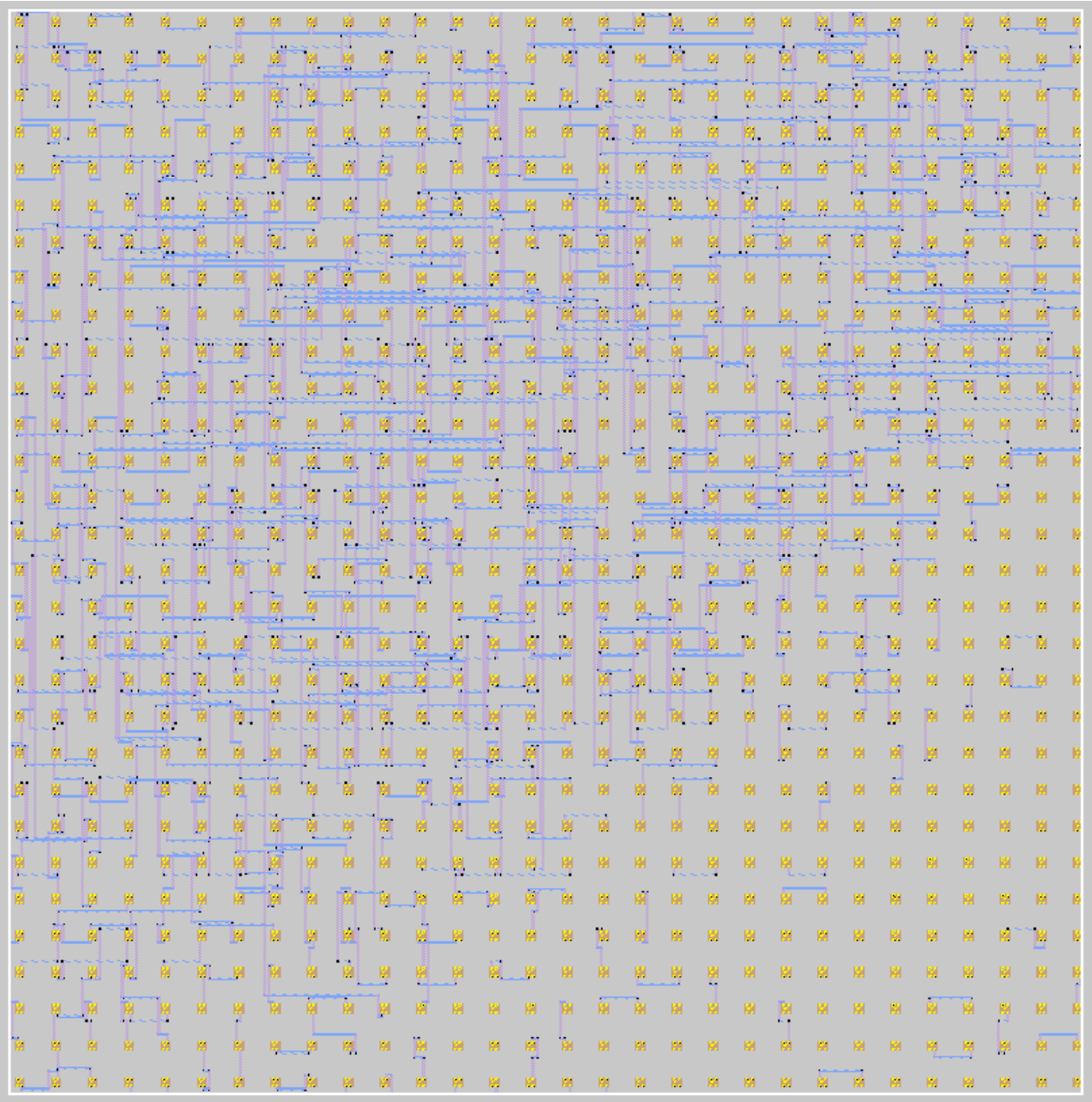


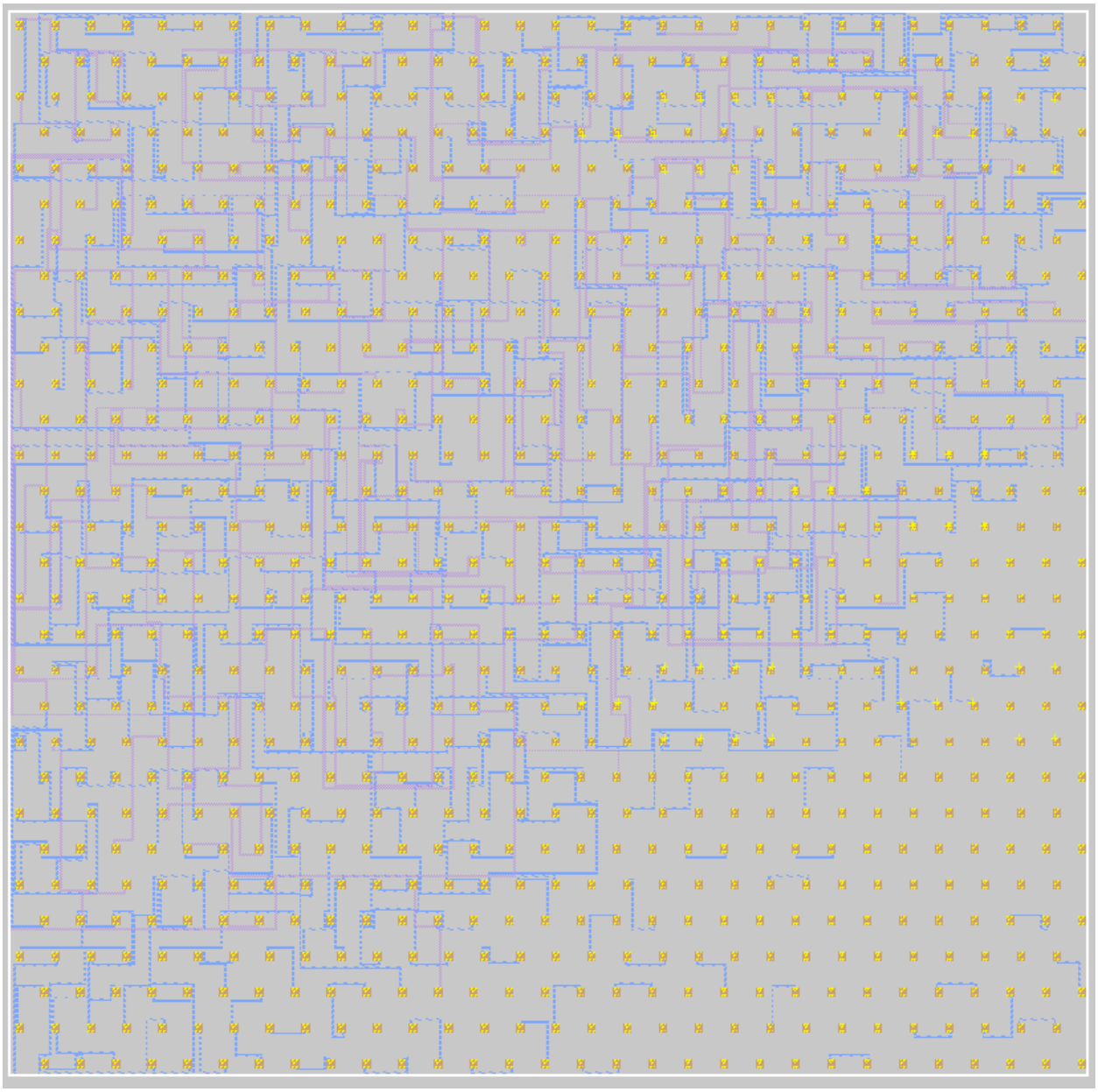
BENCHMARK 7



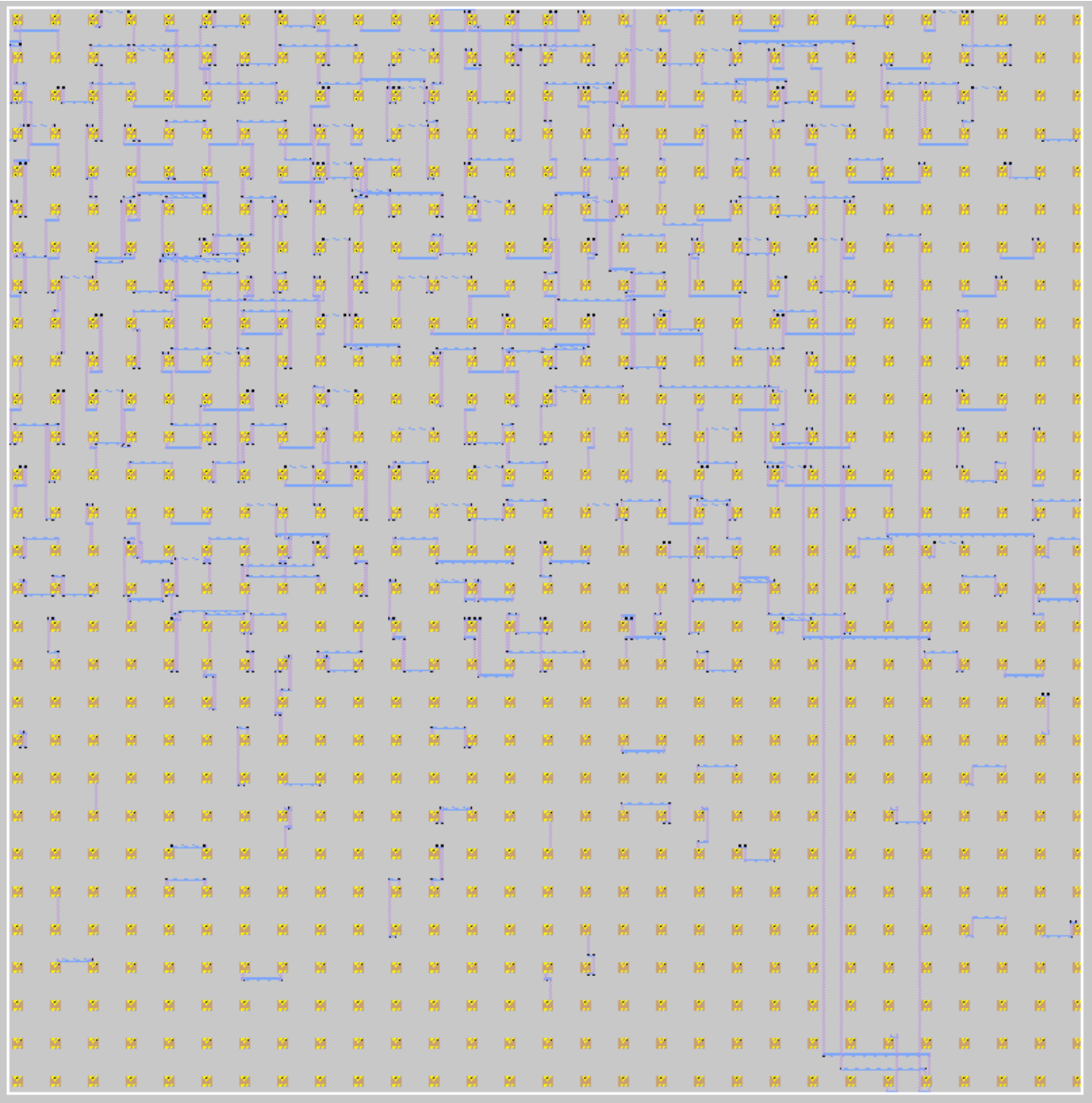


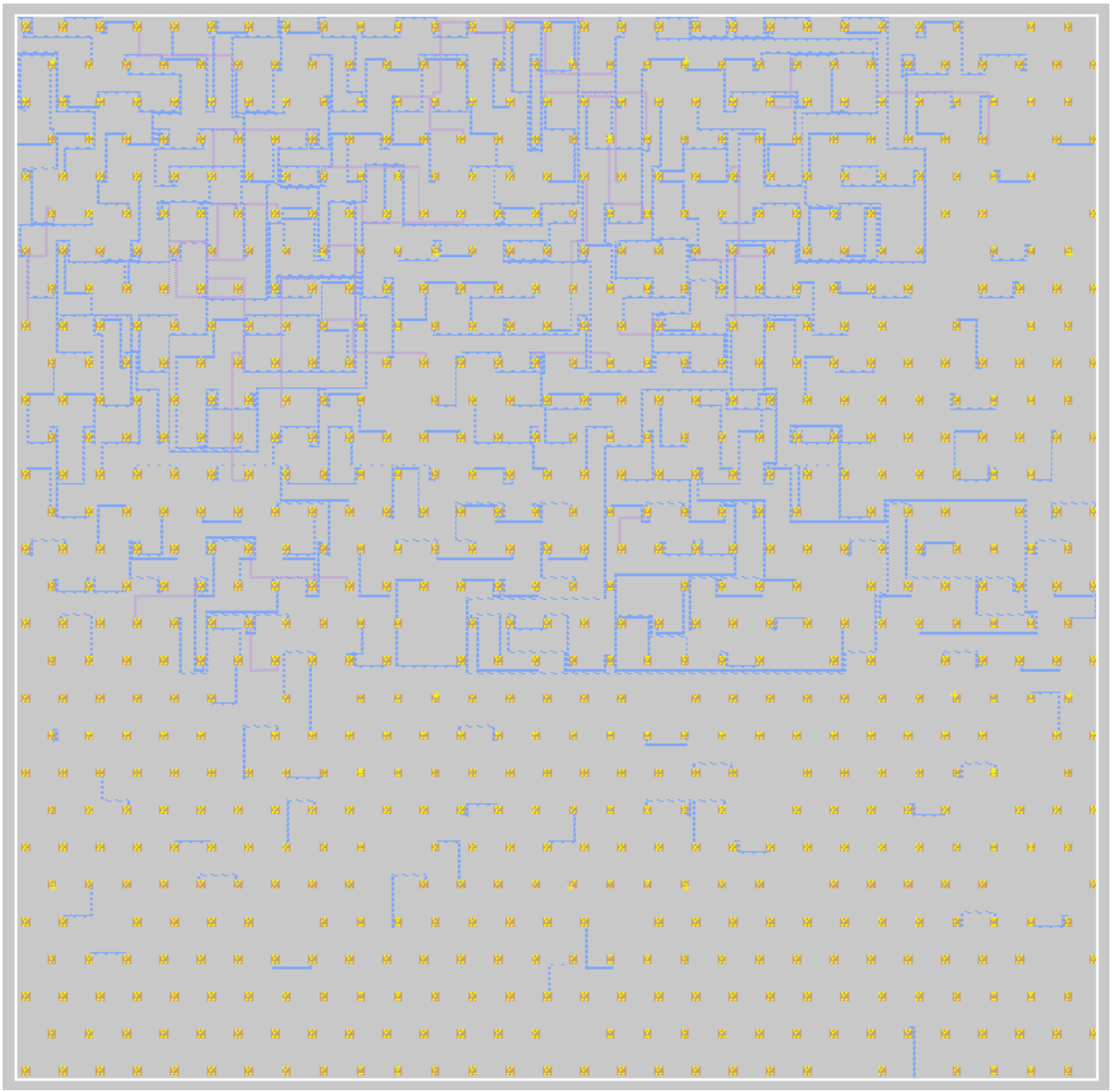
BENCHMARK 8





BENCHMARK 9





BENCHMARK 10