ECE1387 – Assignment 1 – FPGA Maze Routing

Kevin Murray – 996088175

October 12th, 2012

File locations

Executable: /autofs/seth.eecg/nfs.eecg.homes.extgrad/kmurray/ece1387/assignment1/kroute

Source: /autofs/seth.eecg/nfs.eecg.homes.extgrad/kmurray/ece1387/assignment1/SRC

Instructions on running

To run routability driven router only:

```
$ ./kroute -c <circuit_name>
```

To turn on interactive graphics::

```
$ ./kroute -c <circuit_name> --interactive_graphics
```

To turn on reservoir optimization:

```
$ ./kroute -c <circuit_name> --opt_reservoirs
```

Also see:

```
$ ./kroute –help
```

Benchmark Results

Table 2. Benchmark Results

	Total # Segs	**Total # Segs at Min W**	**Default Phi**	**Optimized Phi**	**Min W**
fcct1_12	63	69	48.3	46.5 (-4%)	4
fcct2_12	414	414	336	305.4 (-9%)	6
fcct3_12	4328	4328	3758.6	3647.6 (-2%)	22
fcct4_12	10977	9619.2	9597.9	9402.3 (-2%)	28

Flow Description

The implimented router is based around the Pathfinder negotiated congestion algorithm. It consists of a cost drivn maze router (Lee's algorithm) used to route individual nets, being run inside a larger outer loop that handles congestion.

The algorithm operates on a routing resource graph generated from the circuit description of grid size and channel width. The wires are stored as an array, with pointers to switchblocks and pins accessible from each wire. The switchblocks are used to describe the connectivity between nets. Each pin is associated with a CLB in the design. The connectivity between nets described in the input file is stored in an array of nets. Each net includes the source and destination pins, and if the net is routed, it also stores a list of pointers to the used routing wires.

The main net routing routine is try_route_net(), found in maze_route.c. It impliments the cost driven maze router largely as outlined in class. One of the key decisions was how to impliment the expansion list. Since the expansion list acts as a priority queue, it is important to be able to perform both fast insertions and deletions. This requirement makes an array or linked-list structure unsuitable. Two possible data structures are a heap, and a binned array. The binned array allows fast insertions and deletions provided it does not become too full, and the cost values are

uniformally distributed. To avoid these restrictions, a heap was choosen. The heap based priority queue is implimented in expansion_list.c. During the expansion step, the incremental cost of each wire is calculated, and the total cost (from source to the current expansion wire) is used to label each wire segment. Traceback selection is simplifed since a disjoint switchblock is assumed. The disjoint switchblock means the choice of the first (target) wire determines the track used to route the entire net. To enable the negotiated congestion aspect of path finder, try_route_net does not attempt to avoid resource overuse.

The larger outerloop that handles congestion is pathfinder_route(), found in pathfinder.c. Following the basic pathfinder algorithm as outlined in class, the router will rip up and re-route each net in the design, so long as overused resources exist. During this process it will update the cost of the node based on its current overuse, and its historical overuse. The cost functions are based on those proposed by Betz et al [1]:

$cost(n) = p(n)*h(n)*r(n)$

The cost function is split into p(n) the present cost (based on the current overuse of the wire), h(n) the history cost (based on the historical overuse of the wire), and r(n) which will be described below. As outlined by Betz et al, these values are tunable by several factors, and the following selections were made achieve reasonable routing performance.

Table 1. Cost Function Tunable Parameters

	Value	**Comment**
H_{fac}	0.5	Constant
$P_{fac\_initial}$	1	Slight initial penalty for overuse
P_{fac_mult}	2	Increasing penalty for overuse
$R_{fac\_initial}$	2	Initial penalty for utilizing two adjacent reservoir wires
R_{fac_mult}	0.1	Decreasing penalty for utilizing two adjacent reservoir wires

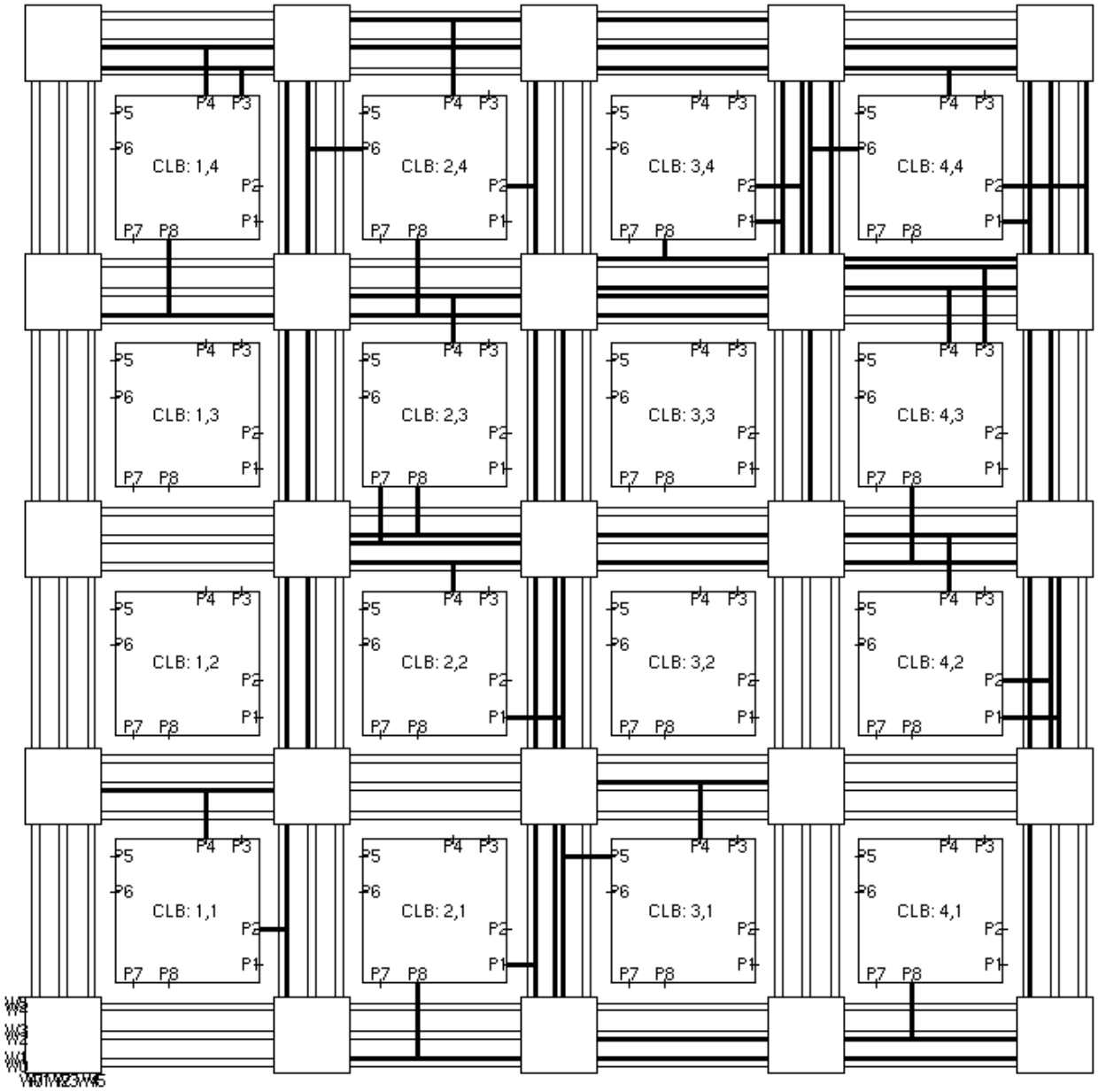
To optimize for phi, the router attempts to maximize the number of unused reservoir segments. The cost function was modified with the addition of an reservoir cost r(n). This is used to penalize tracks odd numbered tracks (which are reservoirs for the even numbered tracks), it is set using the following form, where i is the routing iteration number:

$r(n) = 1.0$ if n is an even numbered track
$r(n) = 1.0 + (R_{fac\_initial})*(R_{fac\_mult})^{i-1}$ if n is an odd numbered track

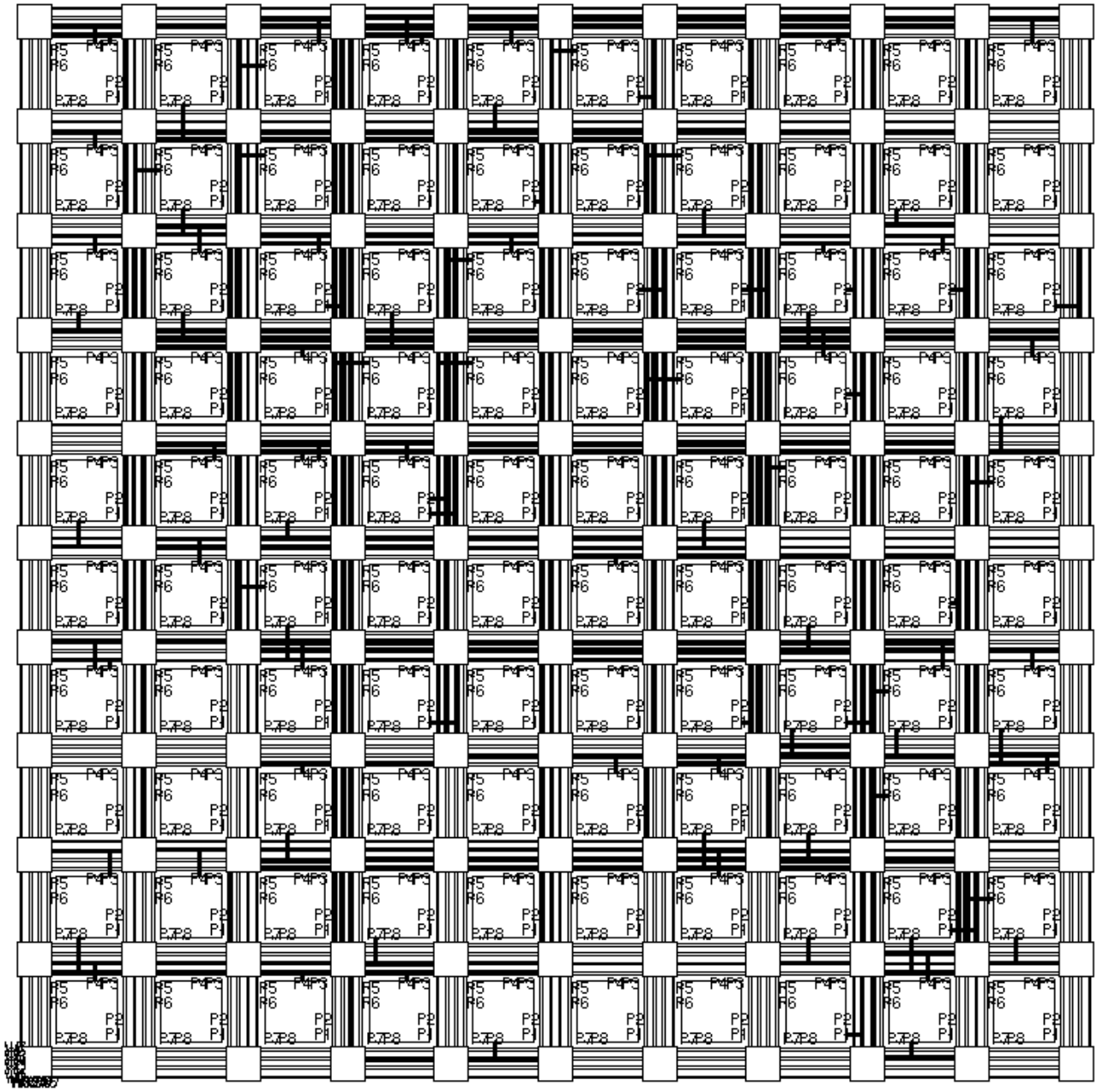
This formulation ensures that r(n) is 1.0 + $R_{fac\_initial}$ (i.e. 3) during the first routing iteration (i = 1), and then decreasing towards 1.0 as the number of iterations increases. This allows the router to initially try avoiding odd numbered tracks (preserving reservoir wires), but in later iterations stops penalizing them to ensure design routability.

References

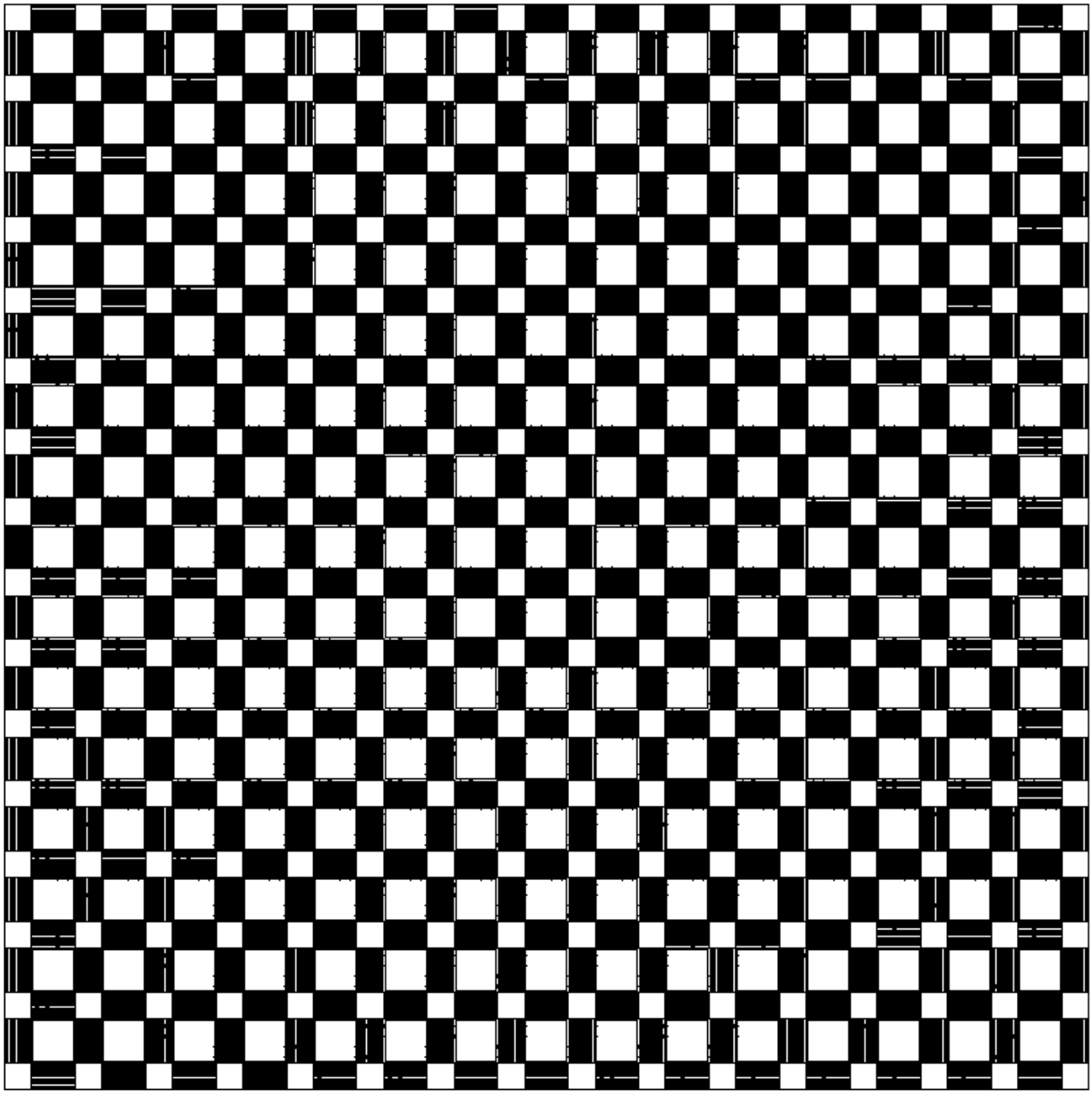
[1] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, February 1999.


CLB: 1,4
CLB: 2,4
CLB: 3,4
CLB: 4,4
CLB: 1,3
CLB: 2,3
CLB: 3,3
CLB: 4,3
CLB: 1,2
CLB: 2,2
CLB: 3,2
CLB: 4,2
CLB: 1,1
CLB: 2,1
CLB: 3,1
CLB: 4,1
P1
P2
P3
P4
P5
P6
P7
P8

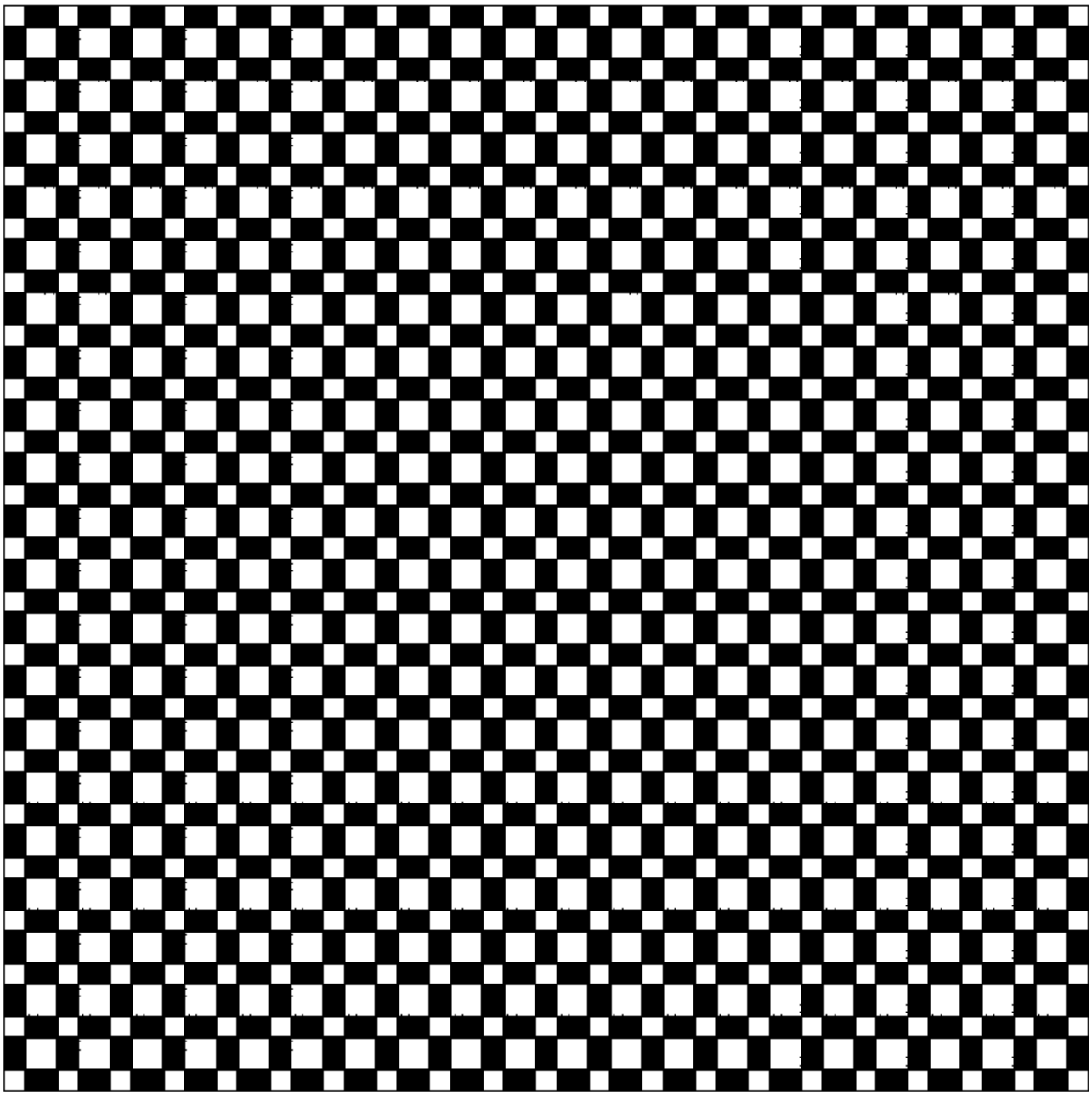
fcct1_12



fcct2_12



fcct3_12



fcct4_12