**1) Files:**

a. New C++ Source Files:

1. bin.h/cpp [ *Class to model a single bin* ]

2. bin2dMap.h/cpp [ *Cell shifting and row/columns of bins* ]

b. Original source files modified:

1. placement.cpp [ *mymain() function is updated* from Windows to Linux]

2. UmfpackHelper.h [ *Added 2 methods to get solver x,y arrays*]

3. graphic.h/cpp [ *EasyGL graphic library* ]

**2) How to compile and Run**:

a. Development Environment:

1. Used g++ on Linux machine, in the Design-Center (not Windows).

2. g++ placement.cpp AnalyticForm.cpp Block.cpp

FixedBlock.cpp Net.cpp NetList.cpp

UmfpackHelper.cpp **Bin2dMap.cpp Bin.cpp graphics.c**

-lumfpack -lamd -lX11 –lm **–o fastPlaceRun**

b. Running the program:

1. ./**fastPlaceRun** *inputFile iterationCount binRowCount binColcount*

2. ./**fastPlaceRun** *spla\_4.ap 20 6 6*

c. GUI Navigation:

1. The initial placement is displayed when the program is started.

2. Click the ‘**Proceed**’ button to display the next iteration results.

3. Click the ‘**Show UnEqual**’ button to display the unequal bin boundaries for the current placement.

4. Color coding:

1. Fixed Cells in RED.

2. Moveable Cells in GREEN.

3. Equal Bins boundary in GREY.

4. Unequal Bins Rows boundary MAGENTA.

5. Unequal Bins Columns boundary BLUE.

**3) High Level Design** :

a. Entry Point call:

1. In the main() function, after the initial solver, we need to create the ***Bin2dMap*** object and only call it’s ***doCellShifting()*** method.

b. Class Description **Bin2dMap** :

1. Initially, all equal bins are created in the Bin2dMap class and stored as rows and columns: vector< vector<Bin\*> > binsRowColVec;.

2. Next we go into the iteration loop, performing all the steps of cell shifting and adding of pseudo pins. Following is the high level flow:

while( not max iteration reached )

{

// Use current solver output to add cells to equal bins.

Bin2dMap.addCellsToBins();

// For each row, calculate enEqual bin boundry.

Bin2dMap.createUnEqualBinsRows();

//For each column, calculate enEqual bin boundry.

Bin2dMap.createunEqualBinsColumns();

//For each movable cell, add pseudo pins and springconstant.

Bin2dMap.addSpreadingForces();

//Call solver with updated matrix A diagonal and bx, by.

Umfpack();

//Clear unequal bin boundary for next iteration.

Bin2dMap.resetBinsInfo();

}

c. Class Description **Bin** :

1. The Bin class represents an individual bin. All information about a single bin is saved in this class: equal/unEqual boundary coordinates, utilization, and vector to store all cell belonging to a bin.

Test case 1:

1. [File: spla\_4.ap] [Size: 18 18] [# FixedCells: 62] [# MoveableCells: 273]
2. Max bin utilization and Total wire length over 20 Iterations:

0.0

5.0

10.0

15.0

20.0

25.0

30.0

35.0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

Max Bin Utilization

Iteration

**spla\_4.ap**

253000.0

254000.0

255000.0

256000.0

257000.0

258000.0

259000.0

260000.0

261000.0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

Total Wire Length

Iteration

**spla\_4.ap**

**4) Test Case Results** :

a. Test case 1:

1. [File: spla\_4.ap] [Size: 18 18] [# FixedCells: 62] [# MoveableCells: 273]

2. Max bin utilization and Total wire length over 20 Iterations:

3. **Output Screen Shots**: [Iteration 0, 7, 12, 20]