MIXED SOFT/HARD-MACRO-CELL PLACER, USER'S GUIDE

I. FILE STRUCTURE AND COMMANDS

The files structure under working directory is:

```
Useful.class
BioComponents.class
BioMechanics.class
BioLife.class
Life.class
XRayApplet.class
XRayApplet.html
Educate.org
<DesignA>
        DesignA-2.org
<DesignB>
        DesignB-2.org
```

Executing command "java Life" under working directory will start the placement procedure. "Educate.org" is a text version configuration file in which current design and a few important parameters are specified. An individual design is stored in a directory. The input design file is named DesignX-2.org. The intermediate placement results and the final layout are all stored in the same design directory with name DesignX-Y.org where Y is a different index number. User only needs to provide "Educate.org" and "DesignA/DesignA-2.org".

1.1 "Educate.org"

A typical "Educate.org" file looks like:

```
AgeFilePrefix           ami33OTCoptAA/ami33OTCoptAA
// AgeFilePrefix        ami49OTCoptAA/ami49OTCoptAA
AgeBorn                 -2
AgeYouth                30
AgeAdult                150
AgeOld                  180
AgeDie                  1000
AgeEternal              2000
Brief                   10
TrialNumber             1
WirePitch               600
WireLayerNumber         2
AspectRatioMaxRate      1.3
AspectRatioMinRate      0.7
```

Lines starting with "// " become comments. "AgeFilePrefix" defines the location and filename prefix of the current design. "AgeBorn" is the index of the input file, which should be a negative number. "AgeYouth", "AgeAdult", and "AgeOld" are the iteration limits for the first three stages of the placement algorithm. "AgeDie" is the index of the final result. "Brief" defines the briefness of the format used to store design files; 10 means most brief; 0 means most detail. "TrialNumber" defines how many times the placer should try with different random initial layouts. "AspectRatioMaxRate" and "AspectRatioMinRate" define the maximum and minimum deviations from the initial aspect ratio for soft cells, respectively.

1.2 "DesignA/DesignA-2.org"

We use the input file of example "ami33OTCoptAA" to show the convention in writing the design files. "// " and "/*"~"*/" pair provide comments.

```
cell ami33OTCoptAA                          the first cell is the chip itself
        type b                              cell type: b=soft chip; B=hard chip; c=soft cell, soft-body terminal;
                                                   d=soft cell, soft-boundary terminal; C=hard cell

        ratio 1.0                           ratio of current cell size to its real size
        orientation n                       hard cell orientation
        size 100000 100000                  current cell X/Y size
        sizeoriginal 10 10                  original cell X/Y size
        position 0 0                        cell position
        terminal P9                         terminal
                terminaldirection o         terminal signal direction: o=output; i=input
                terminaloffset 26600 0      current terminal X/Y offset according to cell left/bottom corner
                terminaloffsetoriginal 26600 0   original terminal offset
                terminalwidth 10000         terminal width
                terminalsilk P9             the net this terminal connects to
```

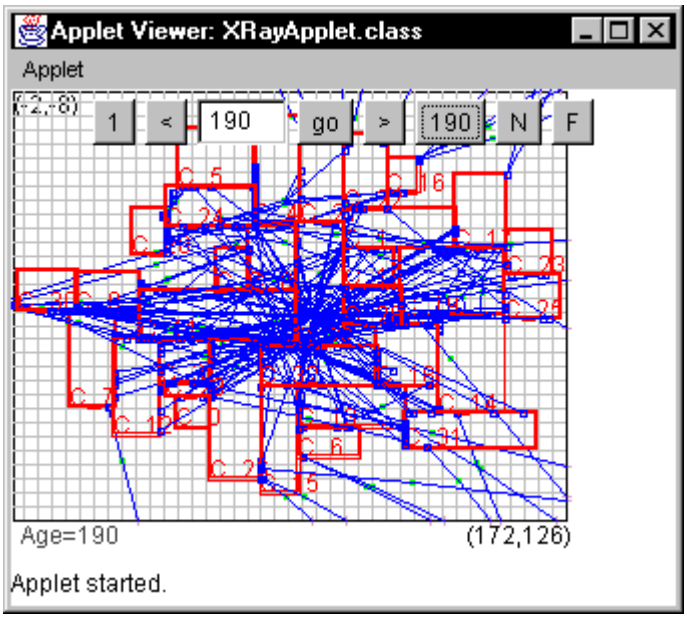
```
                .....
cell C_1                                        cell C_1
        type c
        ratio 1.0
        orientation n
        size 35700 11900
        sizeoriginal 35700 11900
        position 1000 2000
        terminal P_2
                terminaldirection o
                terminaloffset 35000 11900
                terminaloffsetoriginal 35000 11900
                terminalwidth 600
                terminalsilk P30
                .....
.....
silk 327                                        net
        steinerposition 50000 60000             the position of the star of the net
.....
binnumber 38 40                                 X/Y bin number
```

When user gives an input file, all cell ratios should be 1.0.

1.3 Visualization Tool

User can run "appletviewer XRayApplet.html" to pop up a window illustrating the input, intermediate and final layouts if they have been stored in the design directory. For speed purpose, one can choose to suppress intermediate file generation, however this deprives of visualization and debugging for intermediate results.

The window looks like:



User can enter a number and hit "go" to show the result indexed by the number. Button "N" toggles the visibility of nets, and button "F" toggles the visibility of forces. The two numbered buttons reflect the first and the last index. The "<" and ">" can help decrement or increment the index. But if the results are not stored with continuous indexes, e.g., all intermediate results are not stored, then the numbered button may show incomplete information. And entering the index number and hitting "go" always works. Also notice that in Java applet graph, the display axis system is flipped with y-axis, thus on the screeen, (-,-) is north-west and (+,+) is south-east.

II. SOURCE FILES

Source files are java source codes from which the class files are compiled with "javac" command. "Useful.java", "BioComponents.java", "BioMechanics.java", "BioLife.java" and "Life.java" will be discussed in the following. "XRayApplet.java" will not be detailed in this document, but it is fairly easy to comprehend, because its job is merely putting layout information on the screen.

2.1 "Useful.java"

This file defines class “Useful” which includes the following constants and useful methods:

```
int INFINITY_INT=999999999;
double INFINITY_DOUBLE=9.9E+100;
int StringToInt(String InString)
double StringToDouble(String InString)
String DoubleToString(double value)
void WriteALineToFile(String LineField, OutputStream f)
double EuclideanDistanceSquare(int X0, int Y0, int X1, int Y1)
int EuclideanDistance(int X0, int Y0, int X1, int Y1)
```

2.2 “BioComponents.java”

This file defines class “BioComponents” which extends “Useful”. “BioComponents” mainly focuses on defining the data structure, basic functionalities like file-read/write, and a few statistic functions. At the beginning of the file, many parameters are defined and given values. Most of them are constants which can be neither changed by the algorithm nor assigned values by user through “Educate.org”. Whenever user changes the value of these parameters, he or she has to re-compile the “BioComponents.java”. The data structure consists of four main parts: cell, terminal, silk and bin array. The methods in this class are discribed as follows.

void Age(int agenew)	specify current iteration index, ie, “age”
void AgeGrow()	age=age+1
void InitializeCellPosition(int seed)	randomly assign location of cells
void InitializeCellOrientation(int seed)	randomly assign orientation of hard cells
void UpdateAllCells(double newratio)	update ratio of all cells
void UpdateACell(int c)	update a cell’s information
void Explode(double inFactor)	radially spread all cells from the center of the layout
void Read(int age)	read a design file indexed by “age”
void Write()	write the layout of current “age” to a design file
void Educate()	read “Educate.org” file
void Sleep()	cause the program to stop
void Statistic()	layout statistics
ReportTime()	count program run time
ReportLayout()	report layout information

The methods not listed above are all small but frequently used functions.

2.3 “BioMechanics.java”

This file defines class “BioMechanics” which extends “BioComponents”. All algorithmic operations are implemented in this class. The dependencies between operations are shown in the following structure:

void LocateBed(double marginRatio)	determine the layout boundary
void DetermineBin()	determine bin array
void CalculateArgument(int a)	change weights and limits according to iteration index
void CalculateCompleteForce()	calculate all forces
CalculateSilkLength();	calculate wire length
CalculateSilkAttractForceValue();	calculate attractive force on wire based on Hook’s Law
CalculateFillingForce();	calculate density balance force
CalculateBinDensity();	calculate bin density
CalculateBinFillingForce_CLASSICAL();	calculate density balance force generated from each bin
CalculateCellFillingForce();	calculate density balance force of cells
CalculateTerminalSteinerFillingForce();	calculate balance force on terminal and star
void CalculateCellPosition()	calculate new position of cells
void CalculateSteinerPosition()	calculate new position of stars
void CalculateSoftTerminalPosition()	calculate new position of soft terminals
void CalculateSoftBodyTerminalPosition(int c)	calculate position of soft-body terminals
void CalculateSoftBoundaryTerminalPosition()	calculate position of soft-boundary terminals
void CalculateSoftBoundaryTerminalPosition(int c)	calculate position of soft-boundary terminals of a cell
CheckSoftBoundaryTerminal(int c);	check to prevent abnormal position of terminals
DeterminePerimeterBin(int c);	determine 1-dimensional bins on cell or chip boundary
CalculatePerimeterBinDensity(int c);	calculate 1-dimensional bin density
CalculatePerimeterBinFillingForce(int c);	calculate 1-dimensional bin density balance force
CalculateSoftBoundaryTerminal1DForce(int c);	calculate 1-dimensional force of soft-boundary terminals
void CalculateSoftCellShape()	calculate new shape of all soft-cells
void MassageASoftCell(int c)	calculate new shape of a soft-cell
void CalculateHardCellOrientation()	determine whether/which hard cell to change orientation
void CalculateHardCellNewOrientationGain()	calculate the gain of changing orientation of hard cells

void Compaction2D()	2-dimensional compaction
void Compactor(char direction, int maxOver, boolean needWireBalancing)	1-dimensional compaction with wire balancing

2.4 “BioLife”

This file defines class “BioLife” which extends “BioMechanics”. This class contains only one method, “void OneLife()”. The method implements the placement algorithm by organizing methods given in BioMechanics.

2.5 “Life”

This class instantiates an object of “BioLife” and run “OneLife()” of “BioLife”.

2.6 Notice

The “Compaction2D” in “BioMechanics” makes use of a commercial quadratic programming solver called “mosek”. The compactor writes a file (*.mps) and gives it to mosek solver; then reads the solution file (*.sol) and retrieves the compaction results. Mosek program can be obtained at www.mosek.com. Users are encouraged to replace this compactor with their own tools, because we are not very satisfied with the performance of this 2-D compactor. User can compare the results of “AgeOld” and “AgeDie” to see how the compactor changes, and usually ruins, the layout. The compactor used in the old version (pure-hard-cell placer, reported in ICCAD paper) is based on a different algorithm and can give better results, however it consumes more memory and computer time, so it is not adopted in this version. We are developping another compaction algorthm which is supposed to be more intelligent. If user does not care about small overlap between cells, he or she can simply remove the “Compaction2D()” line from “BioLfe”. This will not greatly affect the chip area and the wire length.