

# CS6135 VLSI Physical Design Automation

## Homework 4: Global Placement

Due: 23:59, June. 06, 2017

### 1. Problem Statement

This programming assignment asks you to write a global placer that can assign cells to desired positions on a chip. Given a set of modules, a set of nets, and a set of pins for each module, the global placer places all modules within a rectangular chip. In the global placement stage, overlaps between modules are allowed; however, modules are expected to be distributed appropriately such that they can easily be legalized (placed without any overlaps) in the later stage. As illustrated in Figure 1, for a global placement result with modules not distributed appropriately (Figure 1(a)), the modules cannot be legalized after legalization and detailed placement (modules in the middle bin of Figure 1(b) are overlapped). In Figure 1(c), a more appropriately distributed global placement result is provided, which can be legalized as illustrated in Figure 1(d).

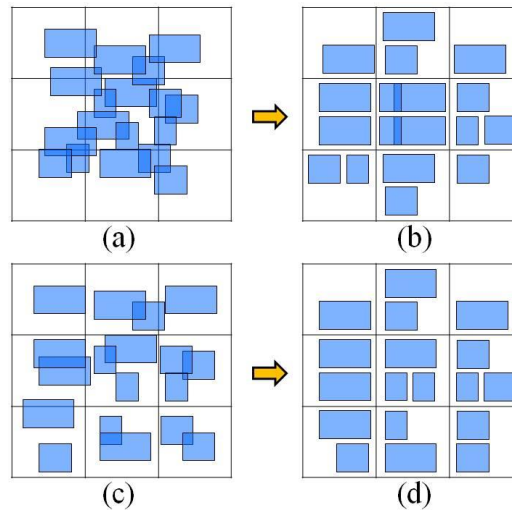


Figure 1. The process of Global Placement

In addition to placing modules appropriately, the objective of global placement is to minimize the total net wirelength. The total wirelength  $W$  of a set of  $N$  can be computed by

$$W = \sum_{n_i \in N} HPWL(n_i)$$

where  $n_i$  denotes a net in  $N$ , and  $HPWL(n_i)$  denotes the half-perimeter wirelength of  $n_i$ . Note

that a global placement result which cannot be legalized is not acceptable, and any module placed out of the chip boundary would lead to a failed result.

## 2. Predefined Data Structure

```
class Module
{
    public:
        /*get functions*/
        string name();
        double x(); // coordinate of the bottom-left corner
        double y();
        double centerX();
        double centerY();
        double width();
        double height();
        double area();
        unsigned numPins();
        Pin& pin(unsigned index); // index: 0 ~ numPins()-1
        /*set functions*/
        // use this function to set the module position
        Void setPosition(double x, double y);
};

class Net
{
    public:
        unsigned numPins();
        Pin& pin(unsigned index); // index: 0 ~ numPins()-1
};

class Pin
{
    public:
        double x();
        double y();
        unsigned moduleId();
        unsigned netId();
};
```

```

class Placement
{
public:
    double boundaryTop();
    double boundaryLeft();
    double boundaryBottom();
    double boundaryRight();
    Module& module(unsigned moduleId);
    Net& net(unsigned netId);
    Pin& pin(unsigned pinId);
    unsigned numModules();
    unsigned numNets();
    unsigned numPins();
    double computeHwpl(); // compute total wirelength
    // output global placement result file for later stages
    outputBookshelfFormat(string fileName);
};

```

The above functions are used to access the data required by the global placement stage. You should use the function **setPosition(double x, double y)** to update the coordinates of modules (to move blocks). At the same time, the coordinates of pins on modules will be updated accordingly and automatically. Note that modules cannot be placed out of the chip boundaries, or the program may not be executed normally.

### 3. Compile & Execution

To compile the program, simply type:

```
make
```

Please use the following command line to execute the program:

```
./place -aux <inputFile.aux>
```

For example:

```
./place -aux benchmark/ibm01/ibm01-cu85.aux
```

### 4. Language/Platform

➤ Language: C or C++

## 5. Submission

You need to submit the following materials in a .tar or a .zip file (e.g., CS6135\_HW4\_101062561.tar.gz. Don't use rar) at the course submission website by the deadline: (1) source codes, (2) executable binaries, (3) a text readme file (readme.txt), stating how to build and use your program, and (4) a report (report.pdf) on the algorithm used in your program. There are examples of readme and report, respectively for you to follow. Also, in your report, please compare your results with the top 3 results from last year.

## 6. Evaluation

This programming assignment will be graded based on the (1) correctness of the program, (2) solution quality, (3) running time (restricted to 2 hours for each case), (4) report.pdf and (5) readme.txt. Please check these items before your submission.

If the global placement result can be legalized, the final solution quality is judged by the total HPWL after the detailed placement stage, which will be shown on the screen as follows:

Benchmark: ibm01		
Global HPWL: 2349680	Time:	13.0 sec (0.2 min)
Legal HPWL: 2531684	Time:	1.0 sec (0.0 min)
Detail HPWL: 2482319	Time:	0.0 sec (0.0 min)
=====		
HPWL: 2482319	Time:	14.0 sec (0.2 min)

However, if the global placement result cannot be legalized, the solution quality will be judged by the following cost function:

$$W \times (1 + \text{"scaled overflow per bin"})$$

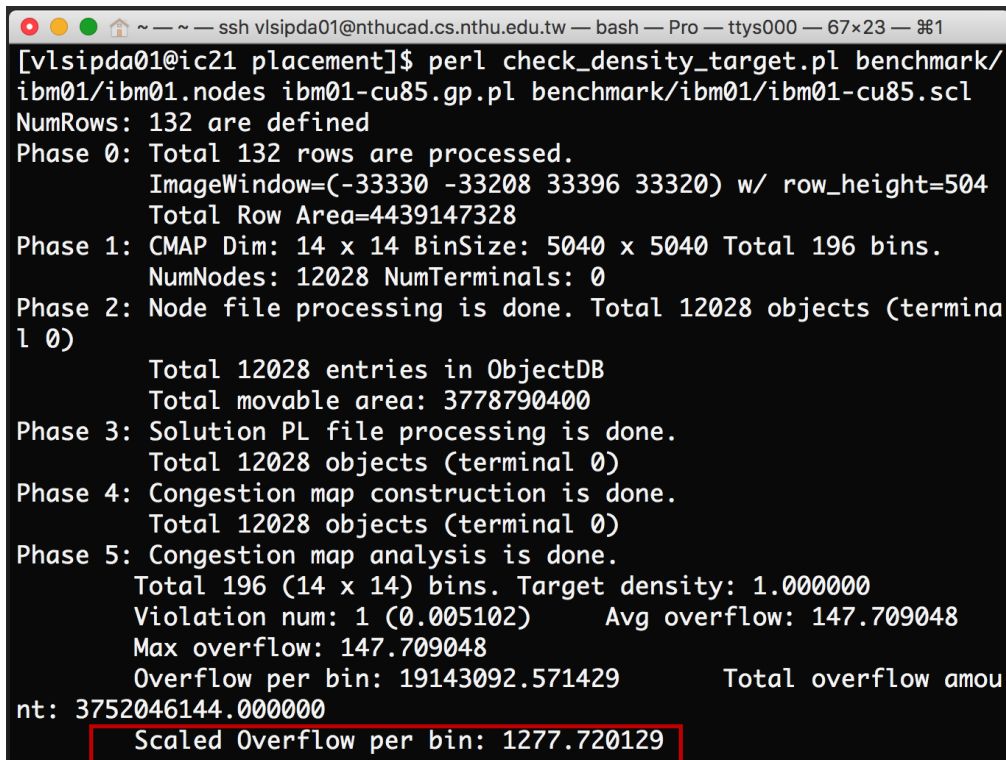
where W is the total wirelength derived from the global placement stage. That is, if you result cannot be legalized by the legalizer, the resultant HPWL will be penalized and become extremely large. The “scaled overflow per bin” can be found by using the following script:

```
perl check_density_target.pl <input.nodes> <Solution PL  
file> < input.scl >
```

For example:

```
perl check_density_target.pl benchmark/ibm01/ibm01.nodes
ibm01-cu85.gp.pl benchmark/ibm01/ibm01-cu85.scl
```

Then “scaled overflow per bin” can be checked on the screen as follows:



```
ssh vlsipda01@nthucad.cs.nthu.edu.tw — bash — Pro — ttys000 — 67x23 — 1
[vlsipda01@ic21 placement]$ perl check_density_target.pl benchmark/
ibm01/ibm01.nodes ibm01-cu85.gp.pl benchmark/ibm01/ibm01-cu85.scl
NumRows: 132 are defined
Phase 0: Total 132 rows are processed.
ImageWindow=(-33330 -33208 33396 33320) w/ row_height=504
Total Row Area=4439147328
Phase 1: CMAP Dim: 14 x 14 BinSize: 5040 x 5040 Total 196 bins.
NumNodes: 12028 NumTerminals: 0
Phase 2: Node file processing is done. Total 12028 objects (terminal 0)
Total 12028 entries in ObjectDB
Total movable area: 3778790400
Phase 3: Solution PL file processing is done.
Total 12028 objects (terminal 0)
Phase 4: Congestion map construction is done.
Total 12028 objects (terminal 0)
Phase 5: Congestion map analysis is done.
Total 196 (14 x 14) bins. Target density: 1.000000
Violation num: 1 (0.005102) Avg overflow: 147.709048
Max overflow: 147.709048
Overflow per bin: 19143092.571429 Total overflow amou
nt: 3752046144.000000
Scaled Overflow per bin: 1277.720129
```

Note that solutions which can be legalized will get **better** scores than those that cannot be legalized.

## 7. Bonus

- ✓ Draw the cell spreading process iteration by iteration and include the snapshots in your submitted report. Remember to turn off this utility in your submitted source code in case of performance reduction.

## 8. Grading

- ✓ 70%~80%: The solution quality (wirelength) and the runtime of each testcase, hidden testcases included
- ✓ 20%~30%: The completeness of your report
- ✓ 5%: Bonus