RAMP: GATE-ARRAY, STANDARD-CELL AND MASTERIMAGE PLACEMENT MANUAL

by

C. K. Cheng and E. S. Kuh

Memorandum No. UCB/ERL M84/71

1 July 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

RAMP: Gate-Array, Standard-Cell and Masterimage Placement Manual

RAMP is an acronym for Resistive Analog Module Placement System. It utilizes the analogy of resistive network to tackle the placement problems, especially the problems of very large scaled integrated circuits. Currently, the system is implemented for the Gate-Array, Standard-Cell and Masterimage Approaches.

The system places the modules automatically. It reads in the specification of the chip, the netlist and the modules with different widths and heights. The user can fix some of the modules. The system uses the objective function of the sum of the squared wiring lengths. It utilizes the network optimization method to distribute the modules into subregions, then, uses spacing process to separate the overlapping modules. In the spacing, the prefixed modules might be shifted from the predefined positions to avoid overlapping.

Currently, RAMP runs on a VAX 11/780 under 4.2 Berkeley UNIX. The HP 2648A is used as the graphics display. The entire system is implemented in C. In order to install the system, user should run **make** in the directories RAMP/sparsity and RAMP/ to compile the program.

Research sponsored by the National Science Foundation Grant ECS-8201580, the University of California, Berkeley, MICRO Program, and by the Semiconductor Research Consortium, grant no. SRC-82-11-008.

RAMP: Gate-Array, Standard-Cell and Masterimage Placement Manual

1. What is RAMP?

RAMP is an acronym for Resistive Analog Module Placement[1-3]. It utilizes the analogy of resistive network to tackle the placement problems, especially the problems of very large scaled integrated circuits. Currently, the System is implemented for the Gate-Array, Standard-Cell and Masterimage Approaches.

1.1. Gate-Array and Standard-Cell Approaches:

The chip has a regular structure. The IO-Pads are placed on the boundary of the chip, while the modules are assigned to an array of rows inside the chip. All modules are assumed to have same height but varying width. The routing area is embedded between the rows and around the periphery of the array. In the Gate-Array approach, rows of transistors are preprocessed. The chip size and routing areas are therefore fixed. In the Standard-Cell approach, chip size and routing areas are adjustable. Thus, routing is different for these two systems. However, the formulation on placement is similar.

1.2. Masterimage approach:

The masterimage approach[4] has the same basic chip structure as that of the Gate-Array. However, the modules to be placed are allowed to have different widths and heights. This method is considered to be intermediate between unconstrainted and constrainted placement.

The masterimage approach releases the constraint on the size of the modules. Thus, the placement of Gate-Array and Standard-Cell approaches are considered as special cases of Masterimage approach.

2. What can RAMP do?

The System places the modules automatically. It reads in data, does assignment and outputs the result.

2.1. Input

The program reads in the following information.

1. Modules: All modules are rectangular with different widths and heights. Some of the modules are fixed by the user. The modules can not be rotated. However, they can be reflected with respect to the x and y coordinates. The pins are fixed on the modules for connections.
2. Netlist: Each net contains a certain list of pins. Nets which contain only one pin or more than thirty pins are deleted by the input subroutine.
3. IO-Pads: IO-Pads are fixed on the boundary of the chip.
4. Chip: Chip has arrays of subregions for the placement of modules.

2.2. Assignment

RAMP employs a the objective function of the sum of the squared wiring lengths. It uses network optimization to divide the modules into subregions, then, uses spacing process to separate the overlapping modules. In spacing, the prefixed modules might be shifted from the predefined positions to avoid overlapping.

2.3. Output

The location and orientation of all the modules.

3. How to use the program?

3.1. Command

To use the program, the user should create two files for the program to read. They are the file (netlist_file) describing the netlist and the file (chip_file) describing the chip specifications. The command "**ramp netlist_file chip_file**" causes the execution of the placement program. The user can define the name of the output file (place_file) by adding place_file after chip_file in the command. Otherwise, the default output file name is "image.P".

3.2. Options

1. Merging: As described in [1-3], after partitioning, we can merge the subregions and do partitioning again to improve the result. However, the execution time would be longer.
2. Plotting: The user can set TOLAYOUT flag for different level of plotting. 0: No plotting, 1: Plot the modules after each level of partitioning, 2: Plot the modules and nets after each level of partitioning, 3: Plot the process of relaxation scheme.

3 beta: beta is a constant used in the relaxation scheme. Based on the value of beta, some modules are fixed at part of the region and the network optimization is done on the other modules to spread the modules over the entire region. It should be set within 0~0.4. We recommend to set the value near 0.25.

4 Debugging: In order to debug, user can set the flag of TOPRINT. 0: No debugging, 1: Print the process, 2: Print the information in partitioning, 3: Print the description of subregions, 4. Print the sparse matrix in network optimization, 5: Print the netlist, 6: Print the information in spacing. User should notice that the printed output might be very large, if the chip is large and TOPRINT is set larger than 1.

5 Interaction: User can set the above options in each iteration of partitioning. Or he can set the options in the beginning and choose no interaction.

4. Input files

4.1. netlist_file

The format of netlist_file is similar to the Berkeley Building-Block Layout System (BBL) input file[5]. The prefixed modules are indicated by the letter 'f' after the string "MOD". Rectilinear modules are allowed; however, they are replaced with the minimum rectangles containing the module by the input subroutine. Based on our network optimization model, any subset of movable modules should connect other modules or IO-Pads.

4.2. chip_file

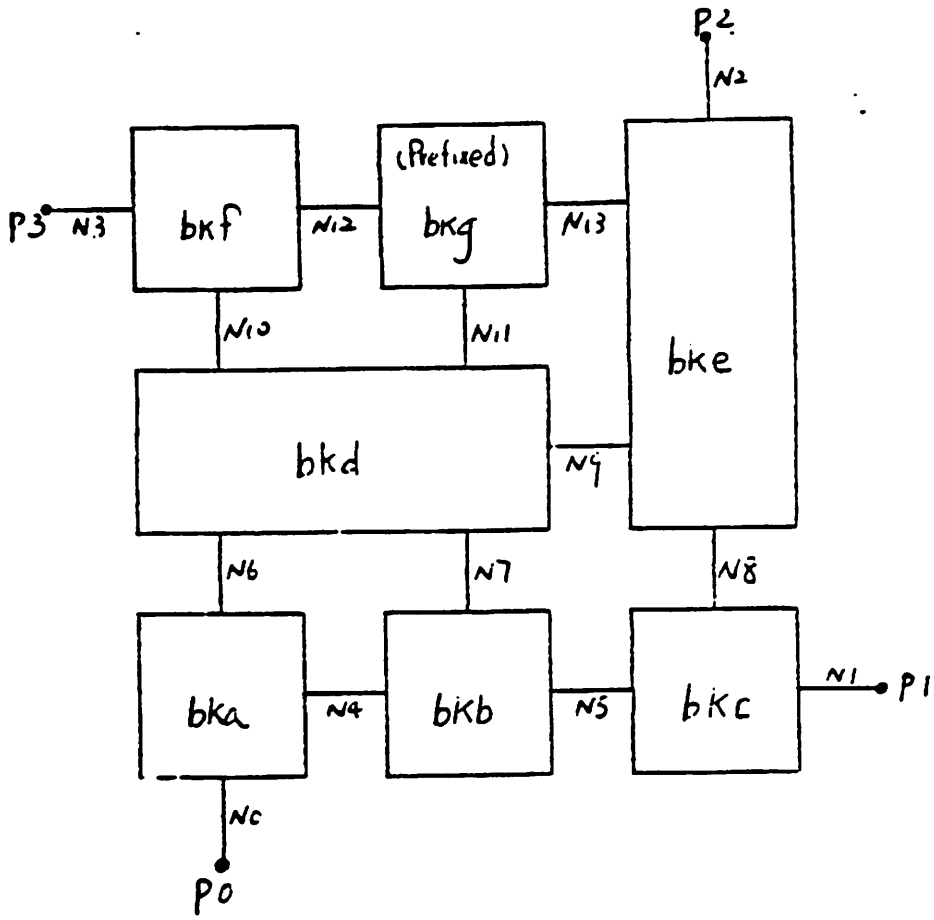
The chip_file describes the chip specifications. The first four lines define the window to be plotted on the screen of HP2648A graphics terminal. The array structure of the chip is described by the left and right x coordinates of each column and the bottom and top y coordinates of each row.

```
x <lx>    /* left x coordinate of the window used in plotting the chip */
X <rx>    /* right x coordinate of the window used in plotting the chip */
y <by>    /* bottom y coordinate of the window used in plotting the chip */
Y <ty>    /* top y coordinate of the window used in plotting the chip */
W <number of columns>    /* This line should be above the lines starting with w */
H <number of rows>    /* This line should be above the lines starting with h */
w 0 <lx0> <rx0>        /* left and right corner of each column */
w 1 <lx1> <rx1>        /* It should be described from left to right */
: :  :   :
h 0 <by0> <ty0>        /* bottom and top corner of each row */
h 1 <by1> <ty1>        /* It should be described from bottom to top */
: :  :   :
```

5. Example

This example has 3 by 3 subregions. The circuit contains eleven modules including four IO-Pads. There are fourteen nets and twenty eight pins. On the graph, "bkg" module is prefixed by the user.

5.1. netlist_file

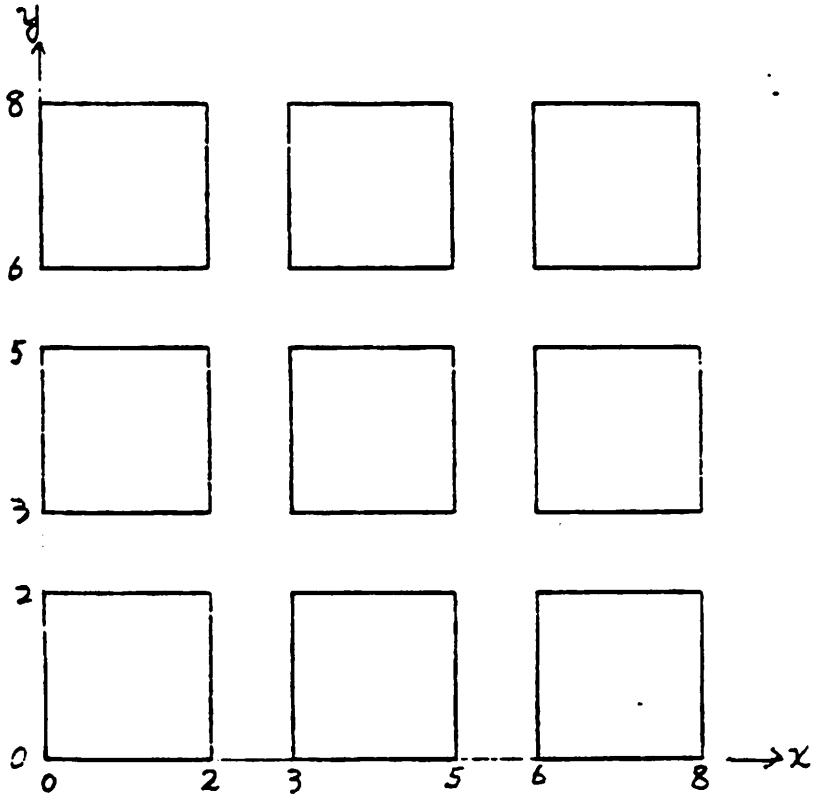



```
SN 14
MOD
0 0
%boun
0
0 0
8 0
8 8
0 8
$
T
1 -1 N0 2 2
9 1 N1 3 2
7 9 N2 0 2
-1 7 N3 1 2
$
DES
ht 1
vt 1
```

```
he 1
ve 1
$
MOD
0 0
bka
0
0 0
2 0
2 2
0 2
$
T
0 1 N0 0 2
2 1 N4 2 2
1 2 N6 3 2
$
MOD
3 0
bkb
0
0 0
2 0
2 2
0 2
$
T
0 1 N4 0 2
2 1 N5 2 2
1 2 N7 3 2
$
MOD
6 0
bkc
0
0 0
2 0
2 2
0 2
$
T
0 1 N5 0 2
2 1 N1 2 2
1 2 N8 3 2
$
MOD
0 3
bkd
0
0 0
5 0
5 2
0 2
$
```

```
T
1 0 N6 1 2
4 0 N7 1 2
5 1 N9 2 2
1 2 N10 3 2
4 2 N11 3 2
$
MOD
6 3
bke
0
0 0
2 0
2 5
0 5
$
T
0 1 N9 0 2
1 0 N8 1 2
0 4 N13 0 2
1 5 N2 3 2
$
MOD
0 6
bkf
0
0 0
2 0
2 2
0 2
$
T
1 0 N10 1 2
2 1 N12 2 2
1 2 N3 3 2
$
MOD f
3 6
bkg
0
0 0
2 0
2 2
0 2
$
T
0 1 N12 0 2
1 0 N11 1 2
2 1 N13 2 2
$
$
```

5.2. chip_file




```
x -1
X 9
y -1
Y 9
W 3
H 3
w 0 0 2
w 1 3 5
w 2 6 8
h 0 0 2
h 1 3 5
h 2 6 8
```

5.3. Execution

Given the input files, user types "**ramp netlist_file chip_file**" to start the execution. The program does process on three phases: Input Phase, Assignment Phase and Output Phase. The information of Input Phase demonstrates on the screen as:

```
|I.INPUT PHASE      ##########################################################
|                        1 User_time, 8 System_time.
|  ====Read netlist from file: Test/netlist_file to BBL database.
|  ====Read chip description from file: Test/chip_file.
```

```
====Transform from BBL database to Array.h structure
    This chip has 11 cells, 4 pads, 14 nets and 28 pins.
    The chip is divided by 3 by 3 subregions.
    It takes 5 level of partitioning process.
====Set the options.
    Default options: No debugging, No merging, beta= 0.25
                     No interaction, Plot graph in the process
    Use the default options? (y/n) [y]:
```

In the interrogation, the user can type a return key to choose the default value as set in the bracket. Or he can type y or n with a return key to set the option. The default options are set no debugging, no merging and no interaction mode. The beta value is set at 0.25. The program will plot the process on the screen, if it is the HP 2648A terminal. Thus the program ask the type of the terminal.

```
    Is this HP 2648A GRAPHICS TERMINAL? (y/n) [y]:
```

If this is not an HP 2648A terminal, the user should response n. Otherwise, the plotting of the graph will generate garbage on the screen.

If the user wants to set options, the program will ask the following questions

```
    Print process data for the debugging? (y/n) [n]:
    value of beta? (0.0-0.4) [0.25]:
    Plot the result? (y/n) [y]:
    Do merging? (y/n) [n]:
    Continue to interact, or not? (y/n) [n]:
```

Then he can set the flags of options. If the value of any flag is set outside the legal value, the program will keep on asking the same question until it is correct.

Finally the program asks the user any change he wants to make. If the options are right, then he can response n. Or, the interrogation process will start again to make sure every option is right.

```
    Want to change the options? (y/n) [n]:
```

At the end of Input Phase, the program will check the format of netlist and chip description to avoid the input of garbages.

```
====Check the format of netlist and chip descriptions.
################################################## END OF INPUT PHASE.
```

In the assignment phase, there are two sections: Initialization Section and Iteration Section.

```
II.ASSIGNMENT PHASE ##################################################
                    19 User_time, 26 System_time.
i.INITIALIZATION SECTION:
```

In the Initialization Section, the program checks the sum of the manhattan lengths and the sum of the squared lengths of the original placement. All the subregions are initialized and the optimization is done.

```
====Check the original placement.
        manhattan length = 14
        squared  length = 14
                  20 User_time, 27 System_time.
====Initialize the subregions.
====Do optimization.
    The result of optimization:
        manhattan length = 46.9767
        squared  length = 99.3697
```

In the Iteration Section, there are two stages. In the first stage, the critical modules are placed. The other modules are then placed in the next stage.

In the first stage, the chip is partitioned into level 4 subregions. the prefixed modules are set and the capacity of subregions are updated. In each iteration of partitioning, the measurements of wiring lengths are checked to indicate the evolution of placement results.

```
ii.ITERATION SECTION:

====Fix predefined modules.
                  26 User_time, 31 System_time.
====Partition the chip to level 4 to place the critical modules.
                  28 User_time, 31 System_time.
  ************************************************************
    Partition to level 0.
        manhattan length = 33.6388
        squared  length = 71.9867
  ************************************************************
    Partition to level 1.
        manhattan length = 32.586
        squared  length = 69.6415
  ************************************************************
    Partition to level 2.
        manhattan length = 38.6464
        squared  length = 84.2181
  ************************************************************
    Partition to level 3.
        manhattan length = 38.6464
        squared  length = 84.2181
```

Then spacing is used to separate the critical modules and fixed them. The program will also ask the user whether to plot the spacing process or not.

```
====Do spacing on critical modules.
                  61 User_time, 38 System_time.
    Plot the Spacing process? (y/n) [n]:
```

Spacing calls for repeated use of Selection of Preferable Direction, Compaction and Decompaction until no overlapping occurs. The maximal number of

iterations is set by ten.

```
        ———Selection of Preferable Direction;
        ———Compaction in y direction;
        ———Decompaction in y direction;
        ———Compaction in x direction;
        ———Decompaction in x direction;
             manhattan length = 64
             squared  length = 328
        Plot the Spacing result? (y/n) [n]:
  ====Fix critical modules.
                          65 User_time, 40 System_time.
```

In the second stage, network optimization is done to partition the chip into last level of subregions. Then, spacing is done to assign all the modules.

```
  ====Partition the chip from level 0 to level 5 to place all the modules.
                          66 User_time, 40 System_time.
     ***********************************************************
        Partition to level 0.
             manhattan length = 19.4581
             squared  length = 22.952
     ***********************************************************
        Partition to level 1.
             manhattan length = 19.9142
             squared  length = 29.6373
     ***********************************************************
        Partition to level 2.
             manhattan length = 19.7426
             squared  length = 29.75
  ====Do spacing on all modules.
                          83 User_time, 43 System_time.
        Layout the Spacing process? (y/n) [n]:
        ———Selection of Preferable Direction;
        ———Compaction in y direction;
        ———Decompaction in y direction;
        ———Compaction in x direction;
        ———Decompaction in x direction;
             manhattan length = 14
             squared  length = 14
        Plot the Spacing result? (y/n) [n]:
  ====Fix all the modules.
                          89 User_time, 44 System_time.
        All modules are found to be fixed.
##################################################### END OF ASSIGNMENT.
```

In the Output Phase, the program checks the result and writes the placement to a file.

```
III. OUTPUT PHASE  ##################################################
                          90 User_time, 44 System_time.
  i. THE RESULT OF PLACEMENT:
             manhattan length = 14
             squared  length = 14
```

ii. OUTPUT TO FILE: image.P:
95 User_time, 47 System_time.
END OF OUTPUT PHASE.

Because the output file is not defined by the user, the placement is written to file "image.P". The format of the output file is the same as the input netlist_file. In this example, the plot of the placement result looks like the first graph.

6. Reference

[1] C.K. Cheng and E.S Kuh, "Module Placement Based on Resistive Network Optimization" IEEE Trans. on Computer-Aided Design pp.218-225 July 1984.

[2] C.A. Desoer and E.S. Kuh, *Basic Circuit Theory*, McGraw-Hill Book Company, 1969.

[3] C.K. Cheng, *Placement Algorithms and Applications to VLSI Design*, Memorandum No. UCB/ERL M84/40 16 May. 1984 Electronics Research Laboratory Univ. of California, Berkeley.

[4] R.L. Donze and G. Sporzynski "Masterimage Approach to VLSI Design," IEEE Computer Mag., pp. 18-25, Dec. 1983.

[5] N.P. Chen, C.P. Hsu, H.H. Chen, E.S. Kuh and M. Marek-Sadowska, *BBL User's Manual*, Memorandum No. UCB/ERL M83/68 4 Nov. 1983 Electronics Research Laboratory Univ. of California, Berkeley.