BBL.2 USER'S MANUAL

by

N.-P. Chen, C.-C. Chen, C.-P. Hsu, H.H. Chen,
E. S. Kuh, and M. Marek-Sadowska

Memorandum No. UCB/ERL M85/2

24 January 1985

BBL.2 USER'S MANUAL

by

N.-P. Chen, C.-C. Chen, C.-P. Hsu, H. H. Chen,

E. S. Kuh, and M. Marek-Sadowska

Memorandum No. UCB/ERL M85/2

24 January 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

BBL.2 User's Manual

Nang-Ping Chen, Chao-Chiang Chen
Chi-Ping Hsu, Howard H. Chen
Ernest S. Kuh, and M. Marek-Sadowska

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, CA 94720

ABSTRACT

BBL is an automatic layout system for placement and routing in VLSI design. The building-block modules are assumed to be rectilinear, and two layers of interconnection are used. The placement system of BBL consists of three major phases : the bottom-up phase to handle the highly connected module pairs and clusters, the top-down phase to deal with the size and the shape of modules, and the trade-off phase to minimize the global connections as well as the overall chip area. The current placement system can only handle rectangular modules. The routing system of BBL includes prerouting analysis, global routing, and detailed routing. The purpose of prerouting analysis is to allocate routing space if the original placement is not desirable. In global routing, a Steiner-Tree-On-Graph algorithm is used to assign each net a specific route without actually embedding it. Nets which belong to a common bus will be assigned the same global route. In detailed routing, channel router and switch-box router are used to do the track assignment. Power and ground nets may have different wire widths, and they will be routed on one layer unless they cross each other. Since modules can be shifted during the routing process, 100% routing completion is always guaranteed.

Currently, BBL runs on a VAX 11/780 under 4.2 Berkeley UNIX. HP 2648A terminal is used as the graphics display, and the final layout will be generated in CIF format. The entire BBL system is implemented in C language, except the channel router, which is written in PASCAL. Many examples from industry have been tested. Experimental results show that the chip area can be reduced by 10-25% with the BBL layout. For an AMI chip with 33 modules, 132 nets, and 440 pins, it takes 69 CPU seconds to finish the placement and 5.5 minutes to complete the routing.

BBL.2 User's Manual

Nang-Ping Chen, Chao-Chiang Chen
Chi-Ping Hsu, Howard H. Chen
Ernest S. Kuh, and M. Marek-Sadowska

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, CA 94720

1. What is BBL ?

BBL is an abbreviation for the Berkeley Building-Block Layout System. It can be used as an automatic tool to generate the layout of integrated circuits. The design style of building-block layout has the following features:

(1) It uses library cells or user-designed macros as the building-block modules.

(2) Each module may have terminals along its boundary.

(3) All the terminals with the same net name should be connected together.

(4) The objective of the placement and routing is to minimize the layout area.

(5) 100% routing completion can always be achieved.

This approach has a wide application in the random-logic custom chip design. It can also be applied in a hierarchical design where BBL is used as the layout tool on each level.

2. What can BBL do ?

BBL can generate automatic placement and routing of integrated circuits, provided the following conditions are satisfied :

(1) All the modules are convex rectilinear polygons. Note that the current placement system restricts the modules to be rectangular.

(2) Each module may have terminals on its boundary. Terminals with the same net name are to be connected together. While the positions for terminals along the chip boundary can be shifted according to the size of the chip, their relative positions will not be changed. All other terminals are fixed on the boundary of their parent modules.

(3) All the modules are treated as blockages. No wires are allowed to cross the modules.

(4) Pre-placement is not allowed in the BBL placement system. However, since the output of the placement is a routing textfile, the placement system and routing system can be used separately. That is, the coordinates of modules can be specified by the user without using the the completely automatic placement system. Modules should not overlap each other, and all the bottom level modules must be inside the boundary of the top level module.

(5) Two layers are available for interconnection.

(6) Four design-rule parameters should be specified; namely, horizontal (vertical) track spacing, and horizontal (vertical) edge clearance.

The current version of BBL does not allow any prewiring before the automatic routing process. However, interactive routing and wire modifications can be done after automatic routing.

3. The placement system of BBL

The placement system of BBL is called *PARADE*. It consists of three major phases: the bottom-up phase, the top-down phase, and the tradeoff phase.

The bottom-up phase starts with the module-wise connectivity analysis. Based on this result, the pairing process is activated to pair the highly-connected modules. The pairing process continues until some threshold value is reached, that is, when connectivity is no longer considered vital. The outcome of this process is a set of clusters, module-pairs, and single modules. For each highly connected module-pair, a set of potentially good wiring patterns is generated. The information will be used in the next step, i.e. the cluster placement. Its

objective is to arrange the modules of each cluster or module-pair to form supermodules which optimize the usage of space and the distance of connections. During this step, pins of different modules are aligned, relative positions of the modules in each cluster are determined, and wiring areas are allocated for both the local nets and global nets. The supermodules can be of any rectilinear shape, but the cluster placement tends to simplify them. For the single modules, local wiring areas are estimated and attached to their peripheries. This estimation is based on a simple probabilistic model which calculates the average local wiring area irrespective of the placement and routing algorithms used.

The input of the top-down phase is a set of supermodules and single modules with estimated local wiring area attached. From now on, we simply call them "blocks". The top-down process partitions the blocks into several coarse rows of cells on the chip. The cell in each row can be a single block, a set of merged blocks, or a piece of a dissected block. The goal of this phase is to minimize the maximum difference of the row lengths while keeping the maximum difference of the cell heights in each row sufficiently small. A novel geometric bandwidth minimization is devised to make the maximum difference of the cell heights in each row sufficiently small. The minimization of the bandwidth is achieved by orienting, merging, and dissecting the blocks while keeping the number of dissected blocks and merged blocks as small as possible. On the other hand, the minimization of the maximum difference of the row lengths is achieved by an efficient multi-way set partitioning algorithm where repeated differencing and backtracing are employed. The number of rows to be partitioned is determined by the estimated chip area and the sum of the horizontal dimension of the cells after bandwidth minimization.

The goal of the tradeoff phase is to assign "blocks" properly in the generalized row structure such that the dissected modules are restored, and the global connection lengths are minimized. This phase begins with an initial arrangement where rows of cells are placed and all the constraints are satisfied. The dissected blocks are restored in the initial arrangement. Then, an iterative process is activated to do the global paths assignment and the detail placement. The spacings between blocks are determined and the difference of the row lengths is kept

as small as possible by using pair-wise swappings. Finally, the horizontal global tracks are assigned and the rows are vertically compacted.

After all the internal modules are placed, we start to place the I/O pads subject to the sequence constraints specified in the placement textfile. The I/O pads are free to be placed anywhere; however, their sequence on the boundary of the chip must obey the given constraints. This placement system is completely automatic and pre-placement is not allowed. However, the user may change the coordinates of the modules afterwards by editing the routing textfile.

4. The routing system of BBL

The routing system of BBL is called *ROSE*. It can be divided into three parts: prerouting analysis, global routing, and detailed routing. The new features of this routing system are power/ground routing and bus routing.

At the beginning of routing process, a set of "bottlenecks" is generated. Bottleneck is defined as a region between the parallel edges of two neighboring modules. It is a critical region where congestion is most likely to occur. Bottlenecks are very important for the whole routing system. As modules are shifted, the structure of some bottlenecks will be changed.

The prerouting analysis will estimate and allocate the routing space needed, if the initial placement is not desirable. It assumes that the interconnection for each net will be done inside the smallest rectangle which encloses all of its terminals. The probability for a bottleneck to be passed through by this net is then calculated, and the sum of probabilities over all nets is the expected routing density in the bottleneck. The number of tracks needed will be the smallest integer larger than the expected density. Modules are then shifted to allocate the routing space.

The next step is global routing. The purpose of global routing is to assign each net a wiring path without actually embedding it. A global routing graph is generated by representing each bottleneck by an edge. The weight for each edge is defined as follows :

$$\text{edge weight} = A * L + B / C^{N+1}$$

where L is the length of the active bottleneck region, N is the number of available tracks, and A,B,C are the parameters specified by the user. If a shortest path is desired, the length factor "A" should be large to dominate the edge weight. If the chip area and routing congestion are of primary concern, the congestion factors "B" and "C" should be large to avoid allocating extra space. According to our experiences, a combination of A=1, B=50, and C=2 tends to give the best result.

A Steiner-Tree-On-Graph algorithm is then applied on the global routing graph to find the minimum weighted tree which connects all the terminals in a net. The net ordering is determined by the available routing space. The net with less routing space will be routed first. For those nets which belong to a common bus, they will be assigned the same global route. After all the nets have been assigned their routes, we get a better estimation of the routing space needed. The required number of tracks in each bottleneck is equal to its maximum routing density, and a compaction process will be done to remove redundant routing space. After the compaction, the minimum chip size is obtained by the global router, though it might be increased later in detailed routing.

Two detailed routers are used in BBL. One is the channel router, and the other is the switch-box router. The channel router is suitable for the routing problem in a rectangular region with fixed terminals on two opposite edges and floating terminals on the other two edges. The switch-box router, on the other hand, can handle any rectilinear region with fixed or floating terminals. It is not as efficient as the channel router, but it is more flexible. In BBL, the active region of a bottleneck is routed by the channel router, and all other regions are routed by the switch-box router.

Power/ground nets are given higher priorities during the routing process. In global routing, the wire widths for power/ground nets will be calculated after their global routes are found. In detailed routing, the channel router and switch-box router use preprocessors to route the power/ground nets on one layer if

possible, and postprocessors to make jumpers for those signal wires which cross the power/ground nets.

Both the channel router and the switch-box router will return a request for extra space if the given routing area is not sufficient. Some modules will then be shifted to allocate additional space for routing. A 100% routing completion can thus be guaranteed, while the increase in chip size is kept as small as possible.

5. How do you enter input data ?

5.1. A placement input textfile

The input format of PARADE is very similar to that of ROSE. The details can be found in Appendix B.

5.2. A net-list routing input textfile

This is the standard input for ROSE. It includes the description of modules, terminals, design rules, and the net list information.

Two levels of modules are used. The top-level module, which encloses all the modules on the bottom level, is usually the chip boundary. The bottom-level modules, which are treated as blockages, are regular modules.

A terminal must be on the boundary of a module. For terminals on the chip boundary (usually I/O pads), their positions will be moved proportionally when the chip boundary changes during the routing process. Every terminal must have a routing direction, which should point toward the routing region. Power/ground terminals may have different widths. The width of the source terminal should be equal to the sum of the widths of sink terminals. If the power terminal is on a horizontal edge, the width will grow leftward. If the terminal is on a vertical edge, the width will grow downward. The distance between the power/ground terminal and its neighboring terminal (or the corner of its parent module) should be enough to cover the wire width and the design rule spacing.

Four parameters of the design rules should be specified. The horizontal (vertical) track spacing is the minimum spacing required between two horizontal

lines. The horizontal (vertical) edge clearance is the minimum distance required between a horizontal wire and a horizontal boundary segment. Currently all the design rule parameters must be set as 1, so all the coordinates should be scaled down (i.e., divided by the track pitch) before they are used as the input.

A detailed description of the input format is in Appendix C. Theoretically, there is no limit on the number of modules and terminals. However, to make BBL run more efficiently, it is recommended that the input data be limited to 50 modules, 1000 terminals, and 1000x1000 chip size.

5.3. A CIF input file

The user can enter the input data by using the interactive graphics editor KIC. In fact, any graphics editor will do as long as the CIF file generated contains the following layers:

TRM - symbolic layer for terminals
BND1 - symbolic layer for the chip boundary
BND2 - symbolic layer for modules

The chip boundary is a rectangular box which contains all the modules. A module is represented by a box or a rectilinear polygon, and terminals are represented by boxes. The center of a terminal box must be on the boundary of its parent module. Each terminal has a label, and the lower left corner of a label should be inside the terminal box. The spacing between the center of terminals and between the center of a terminal and the corner of its parent module should be equal to multiples of the minimum track spacing.

The CIF input file can be transformed into the standard ROSE input format by using the CIF2ROSE command. The new file generated will then be the netlist input file for ROSE.

6. What is the output of BBL ?

The output generated by PARADE is a routing input textfile. The coordinates of the modules are specified. The locations of the terminals are updated if

rotations and/or reflections are performed on the parent module. The size and the shape of the boundary of the top level module are modified. The I/O pads are reassigned subject to the sequence constraints specified in the placement input textfile. The spacings between I/O pads are calculated proportionally to the spacings given by the input.

The routing pattern generated by ROSE is written into a data base textfile, whose name is specified by the user at the beginning of the program. The format of this output file is described in detail in Appendix D. The user can look at the final placement and routing by using the LOOKDB command. A CIF file can also be generated by the CIFGEN command. Then the interactive graphics editor KIC can be incorporated to do the interactive routing or modification. A final plot can be obtained by using the CIFPLOT. (Both KIC and CIFPLOT are in the Berkeley VLSI Tools package.)

7. Appendix

7.1. Appendix A : Commands and application programs

7.2. Appendix B : Input format for BBL placement

7.3. Appendix C : Input format for BBL routing

7.4. Appendix D : Output format for BBL database

Appendix A : Commands and Application Programs

Contents

cifgen(1) - CIF format generator for BBL

cif2rose(1) - translate CIF format to ROSE format

lookdb(1) - database look or dump program

parade(1) - automatic placement system for BBL

rose(1) - automatic routing system for BBL

rose2parade(1) - translate ROSE format to PARADE format

NAME

cifgen – generate a CIF file from BBL database

SYNOPSIS

cifgen [–option [–option] ...] **input_file output_file**

DESCRIPTION

Cifgen is a *CIF format* generator for BBL. It takes BBL database as the input and outputs a CIF file. The actual size of layout is controlled by input parameters. The result can be examined by an interactive graphics editor ***kic*** [1] or the CIF plotter *cifplot* [2]. The options are:

–h (HPterminal)
Display a layout on the HP2648A terminal.

–d (defaults)
Allow you to change default values of geometrical parameters interactively during the program execution. Default values in CIF units are:
1. metal segment width = 300
2. poly segment width = 200
3. contact size = 400
4. terminal size = 400
5. metal to metal separation = 300
6. poly to poly separation = 200

–c (chip)
Generate the whole chip.

–m (module) *module_name*
Generate the specified module only.

–n (number) *max_depth*
Specify how many levels in the hierarchy are to be translated into the CIF file.

–I (input) *text_file*
Input data from *text_file* . A database file will also be created with the name of input_file.

A typical command of using cifgen to generate the layout for the whole chip from our text example test.db is " cifgen -c -n 2 test.db test.cif ".

FILES

BBL/ROSE/CIF/*

NAME

cif2rose – translate a CIF file into the ROSE input format

SYNOPSIS

cif2rose input.cif input.rose

DESCRIPTION

Cif2rose is an input interface between CIF format and BBL format. In order to generate the net list (a standard input format for *ROSE*), the CIF input file must include the following layer definitions:

TRM : symbolic layer for terminal definition
BND1 : symbolic layer for chip boundary
BND2 : symbolic layer for regular module frame

To define a routing problem, the modules and terminals should be specified as follows :

1. Chip boundary is represented by a rectangular box.
2. Regular modules are represented by boxes or rectilinear polygons.
3. Terminals are represented by boxes on the TRM layer. The center of a terminal box must be on the boundary of a regular module.
4. Terminal labels are specified on the TRM layer. The lower left corner of a label must be inside its associated terminal box.

FILES

BBL/ROSE/CIF/*

SEE ALSO

Berkeley VLSI Tools
KIC (cad)[1]
CIFPLOT (cad)[2]

BUGS

This program generates the old input format for ROSE. Modifications must be made to generate informations for power/ground routing and bus routing.

NAME

lookdb – database display routine for BBL

SYNOPSIS

lookdb filename

DESCRIPTION

This program displays all the database information on an HP2648 terminal. The manual for usage can be printed by typing the HELP command. Currently, the following commands are supported by lookdb:

h : help
q : quit
p : change plot flag
n : change print flag
? : print all the signal and module names
s : identify the specified signal
m : identify the specified module
d : display regular modules
dw : display regular modules with default window
da : display chip routing
daw : display chip routing with default window
x : find the name of a module by cursor
xr : find the name of a routing module by cursor
xm : find the name of a regular module by cursor
R : run channel router in the specified routing module
R2 : run 2D router in the specified routing module
W : define new window
f : find input file name
r : read input file
w : write output file
! : escape

FILES

BBL/ROSE/LOOKDB/*

BUGS

For display on terminals other than hp2648a, you must replace BBL/ROSE/LIB/display.c with your own graphics program.

NAME

parade – automatic placement system for BBL (Building Block Layout)

SYNOPSIS

parade

DESCRIPTION

Parade is the placement system for BBL[1]. It is a completely automatic process. No pre-placement is allowed. The modules are restricted to be rectangular and free to rotate and reflect in any orientation as long as the edges are vertical or horizontal. The objective of the placement is to place and orient the modules in an optimal way such that the final layout area, including the interconnection area, is minimized. The boundary of the top level module is determined after all the bottom level modules are placed and the wiring area allocated. The I/O pads are assigned on the boundary subject to the sequence constraints specified in the placement input textfile. The spacings between the I/O pads are calculated proportionally to the spacings given in the input file.

The system will interactively ask user the following questions:

parade
ENTER THE INPUT PLACEMENT TEXTFILE NAME : <file1>
ENTER THE OUTPUT ROUTING TEXTFILE NAME : <file2>
WHOLE CHIP PLOT ? <y/n>

where file1 is the input placement textfile whose format is described in appendix B. ,and file2 is the output routing textfile for ROSE.

DIAGNOSTICS

A new program which can handle rectilinear modules is under development. It will be provided in our next version of BBL-PARADE.

SEE ALSO

[1] Chen, C. C.; Kuh, E.S., "Automatic Placement for Building Block Layout", Proc. ICCAD, 1984, pp. 90-92.

NAME

rose – automatic BBL routing system

SYNOPSIS

rose

DESCRIPTION

Rose is the automatic routing system for BBL[1,2]. In the process of routing, the system may shift functional blocks and compact the layout to achieve 100% routing completion. The terminal positions should be fixed on the boundaries of functional blocks. The I/O pads are represented by the terminals on the boundary of the bounding box. The bounding box may be shrunk or enlarged in size so that it will become the smallest rectangle which encloses all the functional blocks and interconnections. Although the positions of these I/O pads may be changed after the routing, the ratio of the distances between pads will be kept the same. The design rules of wire-to-wire separations, wire-to-edge clearances are specified in multiples of the unit width. Since no additional restriction will be put on the contact-to-contact separation, the user is responsible to specify the wire-to-wire separation large enough to take care of this situation.

This routing system can handle convex rectilinear blocks with arbitrary shape and sizes. No over-the-block routing is allowed. Currently, the system assumes that two layers are available for routing.

A prerouting analysis is equipped with this system. The purpose of this prerouting analysis is to allocate routing space for a given placement based on a simple uniform probabilistic model. The prerouting analysis is not needed if a good manual placement or automatic placement has been done, but it will be helpful if the original placement is not good. The user also has to specify three parameters which control the global routing. If the user is happy with the placement and does not want to change it drastically, then a large *congestion factor B* or *placement adjustability factor C* should be used. If the user cares more about the shortest length connections for all nets, then a large *length factor A* should be used. The system will interactively ask user the following questions:

rose
Enter input file name : <file1>
Enter output database file name : <file2>
Enter length factor for bottlenecks : (default=1)
Enter congestion factor for bottlenecks : (default=50)
Enter placement adjustability factor : (default=2)
Prerouting analysis ? (y/n)
Compaction after global routing ? (y/n)
Final plotting ? (y/n)

The system will generate a file named "debug" under the same directory. This file contains all the bottleneck information for debugging purpose. File1 is the input file whose format is described in appendix C. File2 is the output of the database which can be seen by using "lookdb", or generate the CIF file by using "cifgen".

SEE ALSO

[1] Chen, N. P., "The Routing System for Building Block Layout", Ph.D. thesis, U. C. Berkeley, 1983.

[2] Chen, N. P.; Hsu, C. P.; Kuh, E. S., "The Berkeley Building-Block Layout System for VLSI Design", Proc. International Conference on VLSI, Norway, August 1983, pp. 37-44.

[3] Chen, N. P.; "New Algorithms for Steiner Tree on Graphs", Proc. IEEE ISCAS, 1983, pp. 1217-1219.

[4] Hsu, C. P., "A New Two-Dimensional Routing Algorithm", Proc. 19th Design Automation Conference, June, 1982, pp. 46-50.

[5] Yoshimura, T.; Kuh, E. S., "Efficient Algorithms for Channel Routing", IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, January, 1982, pp. 25-35.

BUGS

An early version of this program was sent to several cooperating companies, who tried our program and gave us feedback. We fixed some bugs and in addition, added new features in this present version, but by no means will this program be perfect. We continue to welcome comments and will improve it in the future versions of BBL.

NAME

rose2parade – convert from ROSE format to PARADE format

SYNOPSIS

rose2parade

DESCRIPTION

Rose2parade will interactively ask the user to enter the PARADE input file to be generated and the ROSE input file to be translated. This program is implemented to help those who already had their own placement and would like to try the new BBL placement for comparison.

FILES

BBL/PARADE/rose2parade

Appendix B : Input Format for BBL Placement

(1) The input text file format

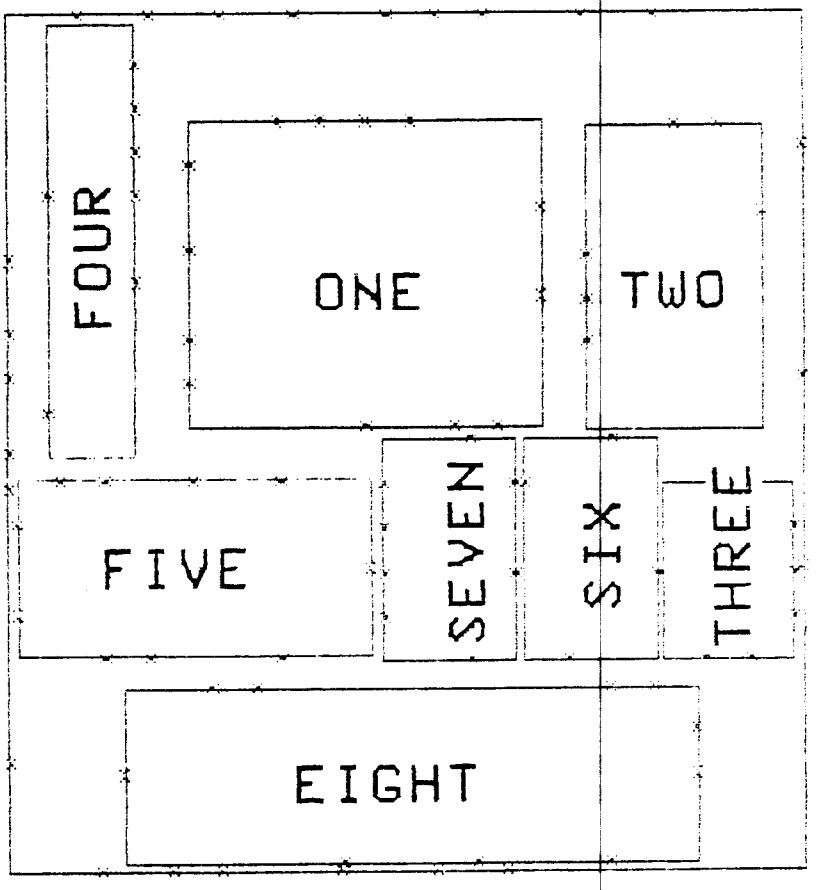
```
<Date>
BBL PLACEMENT TEXTFILE
< # of modules > modules
< # of nets > nets
{top-level module data}
$
{module data at this level}
        :       :
        :       :
        :       :
$
```

(2) The format of module data

```
MOD                         /*top level module*/
0 0                         /*origin coordinates*/
<module name>               /*up to 20 characters*/
<module type>               /*the type of module*/
<dim[X]> <dim[Y]>           /*the dimension of the module*/
$
T                           /*terminals*/
< # of terminals >          /*number of terminals on the module*/
<x> <y> <net> <edge> <type> <layer> <width> <depth> <p/g> <bus>

/*
 * (x,y) ---- terminal coordinates relative to the bottom-left corner
 *             of the module;
 * net   ---- name of the net, up to 20 characters;
 * edge  ---- the edge of the module on which the terminal locates;
 *             i.e. bottom 1, right 2, top 3, left 4;
 * type  ---- 1(fixed), 2(edge fixed), 3(floating);
 * layer ---- the wiring layer that the terminal resides;
 * width ---- the physical width of the terminal;
 * depth ---- the depth of the terminal toward the inside
 *               of the module boundary;
 * p/g   ---- the power and ground flag;
 * bus   ---- the bus flag (see also App.C);
 */
                :       :
                :       :
                :       :
$
```

(3) A sample input file for placement

```
DATE
BBL PLACEMENT TEXTFILE
8 modules
38 nets
MOD
0 0
bound
0
330 230
$
T
24
0 160 N1 4 2 1  1.00  1.00 0 0
0 140 N2 4 2 1  1.00  1.00 0 0
0 130 N3 4 2 1  1.00  1.00 0 0
0 110 N4 4 2 1  1.00  1.00 0 0
0 100 N5 4 2 1  1.00  1.00 0 0
0 30 N6 4 2 1  1.00  1.00 0 0
40 0 N7 1 2 1  1.00  1.00 0 0
70 0 N8 1 2 1  1.00  1.00 0 0
90 0 N9 1 2 1  1.00  1.00 0 0
140 0 N10 1 2 1  1.00  1.00 0 0
180 0 N11 1 2 1  1.00  1.00 0 0
210 0 N12 1 2 1  1.00  1.00 0 0
330 80 N13 2 2 1  1.00  1.00 0 0
330 130 N14 2 2 1  1.00  1.00 0 0
330 190 N15 2 2 1  1.00  1.00 0 0
270 230 N16 3 2 1  1.00  1.00 0 0
240 230 N17 3 2 1  1.00  1.00 0 0
200 230 N18 3 2 1  1.00  1.00 0 0
180 230 N19 3 2 1  1.00  1.00 0 0
160 230 N20 3 2 1  1.00  1.00 0 0
120 230 N21 3 2 1  1.00  1.00 0 0
90 230 N22 3 2 1  1.00  1.00 0 0
60 230 N23 3 2 1  1.00  1.00 0 0
30 230 N24 3 2 1  1.00  1.00 0 0
$
MOD
0 0
ONE
0
80 70
$
T
13
40 0 N26 1 1 1  1.00  1.00 0 0
60 0 N28 1 1 1  1.00  1.00 0 0
70 0 N29 1 1 1  1.00  1.00 0 0
80 30 N18 2 1 1  1.00  1.00 0 0
80 50 N30 2 1 1  1.00  1.00 0 0
50 70 N21 3 1 1  1.00  1.00 0 0
40 70 N23 3 1 1  1.00  1.00 0 0
```



```
30 70 N22 3 1 1  1.00  1.00 0 0
20 70 N24 3 1 1  1.00  1.00 0 0
0 60 N4 4 1 1  1.00  1.00 0 0
0 40 N5 4 1 1  1.00  1.00 0 0
0 20 N2 4 1 1  1.00  1.00 0 0
0 10 N3 4 1 1  1.00  1.00 0 0
$
MOD
0 0
TWO
0
70 40
$
T
6
50 0 N30 1 1 1  1.00  1.00 0 0
70 10 N17 2 1 1  1.00  1.00 0 0
70 20 N37 2 1 1  1.00  1.00 0 0
40 40 N20 3 1 1  1.00  1.00 0 0
30 40 N19 3 1 1  1.00  1.00 0 0
20 40 N18 3 1 1  1.00  1.00 0 0
$
MOD
0 0
THREE
0
40 30
$
T
5
10 0 N17 1 1 1  1.00  1.00 0 0
20 0 N16 1 1 1  1.00  1.00 0 0
30 0 N38 1 1 1  1.00  1.00 0 0
0 20 N18 4 1 1  1.00  1.00 0 0
0 10 N37 4 1 1  1.00  1.00 0 0
$
MOD
0 0
FOUR
0
100 20
$
T
7
40 0 N27 1 1 1  1.00  1.00 0 0
60 0 N33 1 1 1  1.00  1.00 0 0
70 0 N32 1 1 1  1.00  1.00 0 0
80 0 N35 1 1 1  1.00  1.00 0 0
90 0 N34 1 1 1  1.00  1.00 0 0
60 20 N25 3 1 1  1.00  1.00 0 0
10 20 N30 3 1 1  1.00  1.00 0 0
$
MOD
0 0
FIVE
```

```
0
80 40
$
T
10
20 0 N6 1 1 1  1.00  1.00 0 0
30 0 N8 1 1 1  1.00  1.00 0 0
60 0 N29 1 1 1  1.00  1.00 0 0
80 20 N11 2 1 1  1.00  1.00 0 0
60 40 N27 3 1 1  1.00  1.00 0 0
40 40 N26 3 1 1  1.00  1.00 0 0
20 40 N25 3 1 1  1.00  1.00 0 0
10 40 N5 3 1 1  1.00  1.00 0 0
0 30 N1 4 1 1  1.00  1.00 0 0
0 10 N7 4 1 1  1.00  1.00 0 0
$
MOD
0 0
SIX
0
30 50
$
T
4
10 0 N10 1 1 1  1.00  1.00 0 0
30 20 N31 2 1 1  1.00  1.00 0 0
20 50 N27 3 1 1  1.00  1.00 0 0
0 40 N29 4 1 1  1.00  1.00 0 0
$
MOD
0 0
SEVEN
0
50 30
$
T
8
20 0 N9 1 1 1  1.00  1.00 0 0
40 0 N12 1 1 1  1.00  1.00 0 0
50 10 N36 2 1 1  1.00  1.00 0 0
40 30 N35 3 1 1  1.00  1.00 0 0
30 30 N34 3 1 1  1.00  1.00 0 0
20 30 N33 3 1 1  1.00  1.00 0 0
10 30 N32 3 1 1  1.00  1.00 0 0
0 20 N31 4 1 1  1.00  1.00 0 0
$
MOD
0 0
EIGHT
0
40 130
$
T
10
20 0 N12 1 1 1  1.00  1.00 0 0
```

```
40 50 N13 2 1 1  1.00  1.00 0 0
40 80 N14 2 1 1  1.00  1.00 0 0
40 110 N15 2 1 1  1.00  1.00 0 0
30 130 N38 3 1 1  1.00  1.00 0 0
20 130 N37 3 1 1  1.00  1.00 0 0
0 120 N37 4 1 1  1.00  1.00 0 0
0 110 N30 4 1 1  1.00  1.00 0 0
0 30 N28 4 1 1  1.00  1.00 0 0
0 20 N36 4 1 1  1.00  1.00 0 0
$
$
```

(4) Restrictions on input data

The current version of BBL placement has the following restrictions on input data :

* The top level module must be rectangular.
* The bottom level modules are rectangular functional blocks.
* Module type is always 0 (regular).
* The "type" of the terminals on top level module is 2 (edge fixed).
* The "type" of the terminals on bottom level modules is 1 (fixed).
* The "layer" and "depth" of the terminals are set to 1.
* The "width" and "depth" of the terminals are floating number.
* The module and terminal coordinates should be integers.

Appendix C : Input Format for BBL Routing

(1) The input text file format

```
SN <number of nets>
{top level module data}
{design rules}
$
{module data at this level}
        :       :
        :       :
$
```

(2) The format of module data

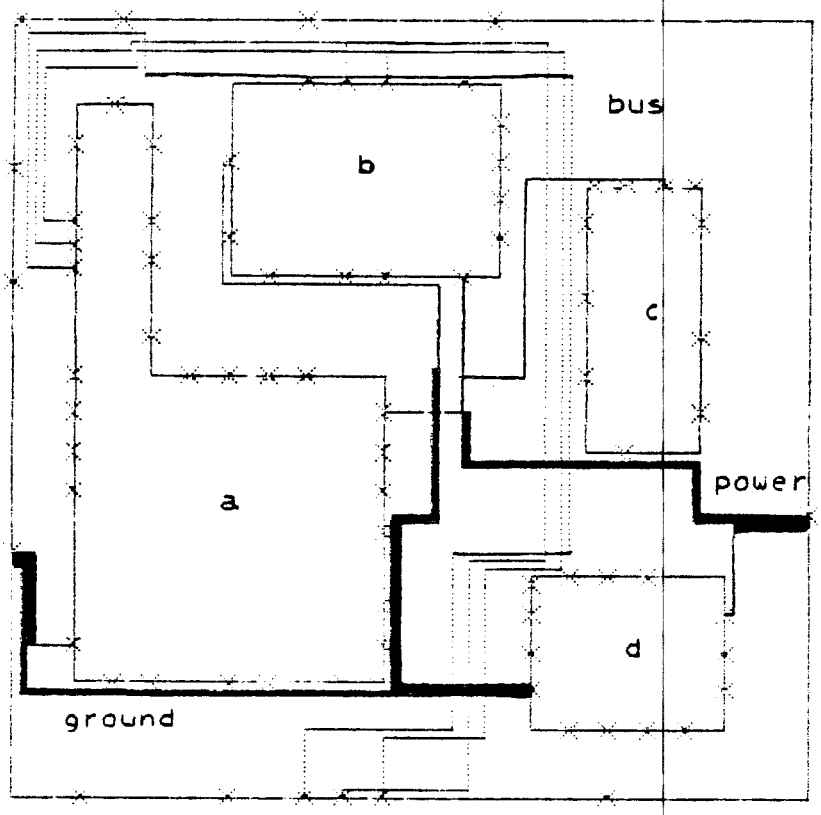
```
MOD                     /*top level module*/
<x> <y>                 /*origin coordinates, all module coordinates are
                          relative to this position*/
<module name>           /*up to 8 characters*/
<module type>           /*1=routing module; 0 otherwise*/
<x1> <y1>               /*corner coordinates of the module in the
                          counterclockwise direction*/
<x2> <y2>
:   :
:   :
$
T                       /*terminals*/
<x> <y> <name> <dir> <type> <p/g> <width> <bus>

/*(x,y) : terminal coordinates relative to the origin*/
/*name of net is restricted to 6 characters*/
/*routing direction : left 0, down 1, right 2, up 3*/
/*terminal type : 2(fixed), other types are for internal use only*/
/*power/ground flag : 1=power/ground, 0=otherwise*/
/*power/ground width : meaningless if power/ground flag=0*/
/*bus number : nets with the same bus number will be assigned the same
  global route*/
/*The specifications of <p/g>,<width>, and <bus> are optional*/
        :       :
        :       :
$
```

(3) The design rule format

```
DES
ht      <horizontal track spacing>
vt      <vertical track spacing>
he      <horizontal edge clearance>
ve      <vertical edge clearance>
$       /* Currently, all the parameters must be 1 */
```

(4) A sample input file

```
SN 29
MOD
0 0
bound
0
0 0
100 0
100 100
0 100
$
T
20 0 u 3 2
40 0 g 3 2
50 0 bus1 3 2 0 1 1
55 0 bus2 3 2 0 1 1
60 0 bus3 3 2 0 1 1
90 0 k 3 2
0 40 ground 2 2 1 4.0
0 75 v 2 2
0 90 o 2 2
10 100 x 1 2
25 100 j 1 2
50 100 s 1 2
75 100 x 1 2
100 45 power 0 2 1 4.0
$
DES
ht 1
vt 1
he 1
ve 1
$
MOD
0 0
a
0
10 85
10 10
50 10
50 50
20 50
20 85
$
T
10 15 ground 0 2 1 1.0
10 35 p 0 2
10 40 p 0 2
10 45 o 0 2
10 50 a 0 2
10 64 bus1 0 2 0 1 1
10 67 bus2 0 2 0 1 1
10 70 bus3 0 2 0 1 1
```



```
10 80 s 0 2
15 10 z 1 2
20 10 b 1 2
30 10 p 1 2
35 10 r 1 2
45 10 w 1 2
49 10 v 1 2
50 15 g 2 2
50 20 f 2 2
50 30 b 2 2
50 35 a 2 2
50 40 d 2 2
50 45 e 2 2
25 50 m 3 2
30 50 l 3 2
35 50 n 3 2
40 50 t 3 2
20 55 n 2 2
20 65 y 2 2
20 70 x 2 2
20 80 w 2 2
15 85 x 3 2
$
MOD
0 0
b
0
65 90
30 90
30 65
65 65
$
T
40 90 bus1 3 2 0 1 1
45 90 bus2 3 2 0 1 1
50 90 bus3 3 2 0 1 1
60 90 l 3 2
30 70 i 0 2
30 80 ground 0 2 1 1.0
35 65 u 1 2
45 65 x 1 2
50 65 j 1 2
60 65 power 1 2 1 1.0
65 70 g 2 2
65 75 y 2 2
65 80 k 2 2
65 85 j 2 2
$
MOD
0 0
c
0
75 45
90 45
90 80
```

```
75 80
$
T
80 45 power 1 2 1 1.0
90 50 f 2 2
90 60 x 2 2
90 75 d 2 2
76 80 c 3 2
80 80 b 3 2
85 80 ground 3 2 1 1.0
89 80 t 3 2
75 55 g 0 2
75 65 i 0 2
75 75 r 0 2
$
MOD
0 0
d
0
85 15
85 35
60 35
60 15
$
T
85 20 l 2 2
85 25 k 2 2
85 30 power 2 2 1 1.0
65 35 e 3 2
70 35 d 3 2
75 35 c 3 2
60 20 ground 0 2 1 1.0
60 25 e 0 2
60 30 g 0 2
60 34 z 0 2
65 15 n 1 2
70 15 m 1 2
75 15 b 1 2
80 15 z 1 2
$
$
```

(5) Restrictions on input data

The current version of BBL has the following restrictions on input data :

* The top level module must be rectangular.

* The bottom level modules are rectilinear functional blocks.

* Module type is always 0 (regular).

* Terminal type is always 2 (fixed).

* The module and terminal coordinates should be integers.

* All the design rule parameters must be 1.

* Power/ground width can be any positive real number. Each power/ground net has one source terminal and several drain terminals. The width of the source terminal must be equal to the sum of the widths of the drain terminals

* bus number must be a positive integer.

Appendix D : Output Format for BBL Database

The output file of ROSE is created by the DBWRITE subroutine. It can be checked directly by using the LOOKDB command, or translated into a CIF file by the CIFGEN command. The first two lines of the output file contain information about the size of each data type. Then 11 types of data are stored in the following order : schip, module, rmpar, geom, gterm, signal, term, srjun, rseg, sroot, and designrl. All records of a given type are dumped consecutively. The output format for each type of record is as follows.

size

Line 1 -
integer, number of schip records in file
integer, number of module records in file
integer, number of rmpar records in file
integer, number of geom records in file
integer, number of gterm records in file

Line 2 -
integer, number of signal records in file
integer, number of term records in file
integer, number of srjun records in file
integer, number of rseg records in file
integer, number of sroot records in file
integer, number of designrl records in file

schip

Line 1 -
integer, module pointer
integer, designrl pointer
integer, signal pointer

module

Line 1 -
integer, length of module name string
******* If non-zero, the next line contains the string.
******* If zero, the next line is (2) below.

Line 2 -
integer, ansmp module pointer
integer, desmp module pointer
integer, sibmp module pointer
integer, mtc term pointer
integer, geop geom pointer

Line 3 -
integer, loc.xy[X]
integer, loc.xy[Y]

Line 4 -
integer, rot
integer, rfl
integer, placg

Line 5 - integer, type
integer, globrt

rmpar

Line 1 - integer, routbnd
integer, chdr
integer, rtflag

Line 2 - integer, param[1]
integer, param[2]

Line 3 - integer, param[3]
integer, param[4]

Line 4 - integer, param[5]
integer, param[6]

Line 5 - integer, param[7]
integer, param[8]

Line 6 - integer, adjx
integer, adjy

geom

Line 1 - integer, gtp gterm pointer
integer, rpar rmpar pointer

Line 2 - integer, lgtp
integer, lbndp

Line 3 - integer, locxy[X]
integer, locxy[Y]

.
.
.
.

Line n - for n size locxy array

gterm

Line 1 - integer, length of name string
******* If non-zero, next line is string
******* If zero, next line is (2) below

Line 2 - integer, loc.xy[X]
integer, loc.xy[Y]

Line 3 - integer, eeg

integer, leg
integer, rdg

Line 4 - integer, placg
integer, clasg
float, pwc
integer, msklvl

signal

Line 1 - integer, length of name string
******** If non-zero, next line is string
******** If zero, next line is (2) below

Line 2 - integer, alls signal pointer
integer, rtls sroot pointer
integer, smp module pointer
integer, trmls term pointer

term

Line 1 - integer, mtc term pointer
integer, stc term pointer
integer, mp module pointer
integer, sig signal pointer
integer, rsp rseg pointer

Line 2 - integer, tnum

srjun

Line 1 - integer, alljr srjun pointer
integer, sljr rseg pointer

Line 2 - integer, locjr.xy[X]
integer, locjr.xy[Y]

Line 3 - short integer, conjr

rseg

Line 1 - integer, widsr
integer, msklvl

Line 2 - integer, type of j0sr (0 = srjun, 1 = term)
integer, type of j1sr (0 = srjun, 1 = term)

Line 3 - integer, allsr rseg pointer
integer, hsr sroot pointer

Line 4 - integer, j0sr record pointer (see line two for type)

integer, j1sr record pointer (see line two for type)
integer, s0lsr rseg pointer
integer, s1lsr rseg pointer

sroot

Line 1 - integer, allseg rseg pointer
integer, alljun srjun pointer
integer, nrts sroot pointer
integer, mp module pointer
integer, shr signal pointer

designrl

Line 1 - integer, htrksp
integer, vtrksp

Line 2 - integer, hegcl
integer, vegcl