BBL USER'S MANUAL

by

Nang-Ping Chen, Chi-Ping Hsu,

Howard H. Chen, Ernest S. Kuh

and M. Marek-Sadowska

Memorandum No. UCB/ERL M83/68

4 November 1983

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

BBL User's Manual

Nang-Ping Chen
Chi-Ping Hsu
Howard H. Chen
Ernest S. Kuh
M. Marek-Sadowska

Department of Electrical Engineering and Computer Sciences
and 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 routing system of BBL can be divided into three parts: 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. In detailed routing, channel router and switch-box router are used to do the track assignment. Since blocks can be shifted during the routing process, 100% routing completion is always guaranteed.

Currently, BBL runs under the 4.2 Berkeley UNIX on VAX 11/780. the whole system is implemented in C language, except channel router in PASCAL. The largest example we have tested so far is the AMI chip with 33 functional blocks, 132 nets, and 440 pins. It takes 5.5 minutes CPU time and 2.5 megabytes core memory to complete the routing.

November 4, 1983

BBL User's Manual

Nang-Ping Chen
Chi-Ping Hsu
Howard H. Chen
Ernest S. Kuh
M. Marek-Sadowska

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-blocks.
2. Each building-block may have terminals along its boundary.
3. All the electrically equivalent terminals should be connected together as a net.
4. The routing problem is described by the net list.
5. 100% routing completion is guaranteed while the chip size is kept as small as possible.

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 an automatic layout of integrated circuits, provided the following conditions are satisfied :

1. All cells are rectilinear polygons.
2. Each cell may have terminals on its boundary. Terminals with the same net number 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 cell.
3. All the cells are treated as blockages. No wires are allowed to cross the cells.
4. Initial placement of functional blocks should be specified by the user. Two blocks should not overlap each other, and all the functional blocks must be inside the chip boundary. The boundary segments can be either horizontal or vertical.
5. Two layers are available for routing.
6. All the wires have the same width.
7. 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 the automatic routing.

3. The routing system of BBL

The routing system of BBL is called *ROSE*. It includes three major approaches: prerouting analysis, global routing, and detailed 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 blocks. It is a critical region where congestion is most likely to occur. Bottlenecks are very important for prerouting analysis and global routing. As blocks 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 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. Blocks are then shifted upwards or toward the right 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 as 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 left, 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. After all the nets have been assigned their routes, we can 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 the detailed routing.

Two detailed routers are used in BBL. One is the channel router, and the other is the switch-box router. 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. 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 the other regions are routed by the switch-box router. Both routers will return a request for extra space if the given routing region is too small. Some blocks will then be shifted to allocate enough space for routing. 100% routing completion can thus be guaranteed, while the increase in chip size is kept as small as possible.

4. How do you enter input data ?

4.1. A net-list input file

This is the standard input for ROSE. It includes the description of modules, terminals, design rules, and the relationship among them.

Two levels of module are used. The top-level module, which encloses all the modules on the bottom level, is usually the chip boundary. The bottom-level module, which looks like a black box, is a regular module. The whole routing region will be divided into several routing modules.

A terminal must be on the boundary of a module. It can be either fixed or floating. If the terminal can only move on one edge, it is called "edge-fixed". All the terminals which are not originally fixed will be called "routing-fixed" after they are routed. Every terminal must have a routing direction, which should point toward the routing region.

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 1.

A detailed description of input format is in Appendix C.

4.2. 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 terminal centers or between a terminal center and its parent module corner should be equal to or 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 net-list input file for ROSE.

5. What is the output of BBL ?

The routing pattern generated by ROSE is written into a data base, which 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.)

Currently, ROSE runs under the 4.2 Berkeley Distribution of VAX UNIX. All the source files are stored under the directory /oc/kuh/ming/BBL/ROSE. HP2648A terminal is used as the graphics display for ROSE, and AED512 color display is used for LOOKDB, KIC, and other interactive graphics editors.

6. Appendix

6.1. Appendix A : Commands and application programs

6.2. Appendix B : Library subroutines

6.3. Appendix C : Input format for BBL

6.4. Appendix D : Output format for BBL

Appendix A : Commands and Application Programs

Contents

NAME

cifgen – generate 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 will also be created.

FILES

/oc/kuh/ming/BBL/ROSE/CIF/*

SEE ALSO

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

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

/oc/kuh/ming/BBL/ROSE/CIF/*

SEE ALSO

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

NAME

lookdb – database look or dump routine

SYNOPSIS

lookdb database-name

DESCRIPTION

This program displays all the database information on an HP2648 or AED512 terminal. The manual for usage can be printed by typing the HELP command. Currently, the following commands are supported by dblook:

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

/oc/kuh/ming/BBL/ROSE/LOOKDB/*

SEE ALSO

dbread(3)

BUGS

Currently, the chip plotting is set to two levels from the top level. For display on the AED terminal, you have to turn on the "lower case" indicator before each command is issued.

NAME

rose – automatic routing system for BBL (Building Block Layout)

SYNOPSIS

rose

DESCRIPTION

Rose is the automatic routing system for BBL[1,2]. It is an all or nothing proposition now. The user is unable to turn off the System unless all the routing is completed. In the process of routing, the System may shift functional blocks in order to guarantee 100% routing and compact the layout. The terminal positions are needed to be fixed on the boundaries of functional blocks. The System is not able to handle the floating terminals currently. The I/O pads are represented by the terminals on the boundary of the bounding box. The bounding box may be reduced or enlarged in size so that it will become the minimal rectangle to enclose all the functional blocks and interconnection. 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 as the input one. The design rules of wire-to-wire separations, wire-to-edge clearences are specified in multiples of the unit width. No additional restriction on the contact-to-contact separation, so the user is responsible to put the wire-to-wire separation large enough to take care this situation.

This routing system can handle rectilinear blocks with arbitrary shape and sizes. No over-the-block routing is allowed. Currently, the System assumes that one metal and one polysilicon layer are available for routing.

A prerouting analysis is equipped with this System. The user has the option to turn it off. The purpose of this prerouting analysis is to allocate routing space for a given placement based on a simple uniform probalistic model. Not very satisfactory results are reported from this analysis because it sometimes distorts the original placement too much and hence put the routing System in a bad starting position. The user has to specify the three constants that control the global routing of the System. If the user is very happy with his placement and does not want to have a big change on it, then he should use a large *congestion* constant. If the user cares more about the shortest length connections for all nets, then he should use a large *length* constant. The System will interactively ask user the following qustions:

rose
ENTER INPUT FILE : <file1>
ENTER DATABASE FILE : <file2>
DEBUG MODE : <y or n>
ENTER LENGTH FACTOR FOR BOTTLENECKS : <integer>
ENTER CONGESTION FACTOR FOR BOTTLENECKS : <integer>
ENTER PLACEMENT ADJUSTIBILITY FACTOR : <integer>
PREROUTING ANALYSIS? <y or n>
FINAL PLOTTING? <y or n>

The System will generate a file named "debug" under the same directory. This file contains all the bottleneck information for the debugging purpose. File1 is the input file whose format is described in ~ming/BBL/ROSE/DATA/input.form. File2 is the output of the System

whose format is described in ~ming/BBL/ROSE/DATA/db.form. Three factors are used to control the global routing. If the user choose the enter the debugging mode, then the System will interactively print and plot the intermediate results. The user can see the plotting of result before the database write which is a time consuming job by answering y to the "FINAL PLOTTING?" question.

FILES

~ming/BBL/ROSE

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", Memorandum No. UCB/ERL M83/10, Electronics Research Laboratories, U. C. Berkeley, Feb. 1983.
[3] Chen, N. P.; "New Algorithms for Steiner Tree on Graphs", Proc. 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.

DIAGNOSTICS

A new program which can handle different wire widths for power and ground nets is under development. It will be provided in our next version of BBL.

Appendix C : Input Format for ROSE

(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

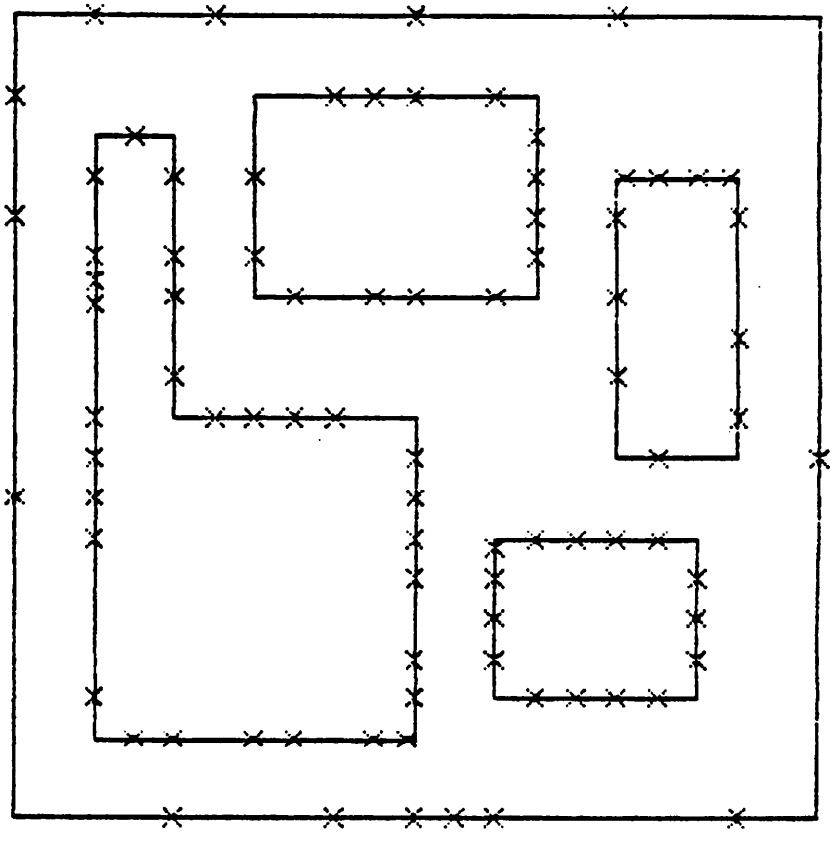
```
MOD                     /*top level module*/
<x> <y>                 /*origin coordinates, all module coordinates are
                          relative to this position*/
<module name>           /*up to 8 characters*/
<module type>           /*1=routing module; 0 otherwise*/
<x1> <y1>          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

/*(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 : 0(floating), 1(edge fixed), 2(fixed), 3(routing fixed)*/
        :       :
        :       :
        :       :
$
```

(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 25
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 m 3 2
55 0 c 3 2
60 0 f 3 2
90 0 k 3 2
0 40 a 2 2
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 h 0 2
$
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 r 0 2
10 35 p 0 2
10 40 p 0 2
10 45 o 0 2
10 50 a 0 2
10 64 v 0 2
10 67 u 0 2
10 70 t 0 2
```



```
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 s 3 2
45 90 y 3 2
50 90 h 3 2
60 90 l 3 2
30 70 i 0 2
30 80 o 0 2
35 65 u 1 2
45 65 x 1 2
50 65 j 1 2
60 65 w 1 2
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 e 1 2
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 a 3 2
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 i 2 2
65 35 e 3 2
70 35 d 3 2
75 35 c 3 2
80 35 h 3 2
60 20 f 0 2
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.

Appendix D : Output Format for ROSE

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, 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