TimberWolf-6.1

Mixed Macro / Standard Cell Floorplanning, Placement and Routing Package

(Building Block, Gate Array, and Standard Cell Circuits)

Yale University

May 1, 1992

Table of Contents

TimberWolf-6.1

Mixed Macro / Standard Cell Floorplanning, Placement and Routing Package

(Building Block, Gate Array, and Standard Cell Circuits)

1. INTRODUCTION

TimberWolf [SecS84][SecS85][Sec88b] is a complete timing driven placement and global routing package applicable to row based and building block design styles. TimberWolf is capable of handling any of the row-based design styles, namely, standard cell circuits, gate arrays and sea-of-gates circuits. In addition, TimberWolf is applicable to circuits containing building blocks or macro cells of any rectilinear shape. Furthermore, the cells may have fixed geometry including pin locations (*hard* macro cells) or the cells may have an estimated area with a specified aspect-ratio range, and with pins that need to be placed (*soft* macro cells). TimberWolf is also applicable to floorplanning problems and may be used to completely place and global route mixed macro/standard cell circuits.

2. FEATURES

Standard cell, macro cell, and mixed macro/standard cell design styles.
Macro cells of any rectilinear shape.
Hard and soft macro cells.
Gate array designs.
X11 (X11R2 - X11R6 inclusive) graphics interface.
Based on new simulated annealing algorithm.
Signal path-based timing driven.
Upper and lower bounds on path lengths.
Wire length calculations are based on actual pin locations.
Flexible pad placement algorithm.
Ability to generate placements close to that obtained from a previous run.
Logical pin swapping and/or gate swapping.
CPU time control via fast/slow option.
Sea-of-gates global router.
N-way timing driven circuit partitioning.
Automatic design flow control.

3. Using This Manual

The following conventions are used throughout this guide:

Boldface	Boldface letters used in the command or pathname examples indicate keywords that must be used literally.
Italic	Italic letters used in the command or pathname examples indicate user- or system-supplied data.
Underline	Underlined text indicates default parameter or setting.
[ ]	Square brackets enclose optional user-supplied data.
< >	Angle brackets enclose specific keyboard key names.
{ }	Curly brackets enclose a set of user choices.

The term *working directory* will refer to the directory of the file system to which the user is currently attached.

4. COMPILATION OF TIMBERWOLF

The distribution tape includes all the files necessary to compile TimberWolf version 6.1. TimberWolf is a collection of programs written in C. The programs should port directly to any machine supporting the C language under the UNIX operating system. Supplemental instructions for the compilation are in the file README included in the top level directory. In order to compile and link the program, the UNIX **make** command is utilized. Since it is a rather involved process to compile and link these modules into an executable, it is recommended to use the UNIX makefile which is supplied. We will outline the necessary steps here. First, set the working directory to be the TimberWolf root directory. Next, type **make** in this directory. This should echo the directions we are about to present. The user must first set the TimberWolf environment variables **TWDIR** and **DATADIR**. **TWDIR** is the pathname of the TimberWolf directory and **DATADIR** is the pathname of the directory where graphics dumps are stored. **TWDIR** and **DATADIR** can be set to their default settings by typing '**source .twrc** <CR>'. This script will also include the TimberWolf bin directory in the search path.

In order to make the build process as flexible as possible, the makefiles have been split into two pieces, the machine independent part or *Ymakefile* and the machine dependent part or *Ymake.macro* file. To compile on different machines, one only needs to set the compile switches found in the *Ymake.macro* file located in the **./pgms/ymake** directory. For your convenience, various *Ymake.macro* files have been included for frequently used configurations. Either edit the *Ymake.macro* file or copy one of the supplied macro files to *Ymake.macro*.

Several compilation switches in the *Ymake.macro* file require further explanation. The conditional compile switch *CLEANUP=-DCLEANUP_C* should normally be defined when running under the UNIX operating system. If for any reason you do not wish to enable the cleanup handler, you may comment out the *CLEANUP* definition in the *Ymake.macro* file. If you wish to install your own cleanup handler instead, see cleanup.h under **./pgms/Ylib/include** and cleanup.c under **./pgms/Ylib/lib** for more details. If you wish to compile a version of TimberWolf without X11 graphics, the *NOGRAPHICS* conditional compile is provided. We DO NOT recommend this unless X11 is **not** supported on your computer. Most workstations support X11, and so the desired method of turning off the graphics is through the use of the **-n** runtime command argument. The macro definitions *XLIB* and *LINKLIB* have been furnished to handle cases where the X11 include files and link library have been moved from their standard places: **/usr/include/X11** and **/usr/lib** respectively. The conditional compile *SYS5* is useful for compilation on system 5 machines. The *DEBUG* option normally includes useful debug code into the final executable. If a faster and smaller executable is desired, the *DEBUG* switch may be commented out. Another compilation switch *NO_FEED_INSTANCES* pertains to the feed-through cell naming convention. By default, feed-through cells are given distinct *instance names*. If you define *NO_FEED_INSTANCES* then each feed will be assigned the *same* name.

After setting the compilation switches in Ymake.macro, return to the TimberWolf root directory and reset the top level makefile by entering '**ymake**'. We are now ready to reset the makefiles for the rest of the system. This is accomplished by typing '**make Makefiles**' on the command line. At this point, we are ready to build the system by entering '**make install_non_yale**'. When the compilation process completes, you will be ready to use TimberWolf.

In designing the TimberWolf system, great pains were made to make the system flexible, portable, and maintainable. Where possible, code is reused through the use of a common library found in **./pgms/Ylib/lib**. All graphics calls to X reside in this library. In addition, most system calls are called

from library functions. In this way, if any problems arise in the compilation process it will tend to be localized in library routines. All character array sizes can be set with the definition **LRECL** which can be found in **./pgms/Ylib/include/base.h**. In addition, the definitions of int, double, and float may all be redefined in this file to accommodate non-32 bit machine architectures.

Note: Never move the TimberWolf tree with a command that does not perserve soft links such as "cp -r". The TimberWolf system will not function if its soft links are destroyed.

5. TIMBERWOLF INPUT FILES

TimberWolf can be executed by issuing the command **TimberWolf** *circuitName*, where *circuitName* is a command line argument specifying the name of the circuit for which the program is to perform placement, global routing, etc. TimberWolf requires the presence of two input files in the working directory: *circuitName.par*, and *circuitName.cel*. The *circuitName.net* file is optional. In addition, the **TWDIR** and **DATADIR** environment variables need to be set to their proper values. Again this can be accomplished by typing '**source .twrc** <CR>' in the TimberWolf root directory.

5.1. THE FORMAT OF THE CIRCUITNAME.PAR FILE

The file *circuitName.par* contains parameter specifications for TimberWolf system. The TimberWolf system is a collection of interacting programs. Instead of having parameter files for each of the individual programs, all TimberWolf programs are controlled through the use of a single parameter file. The parameter file consists of two parts: the design rule parameters and program control parameters. Comments are similar to those found in the csh; they are entered by placing a # in the first column of a line. The format for design rule parameters is as follows:

RULES

layer	*layername*	*resistance capacitance*	{ **vertical** | **horizontal** }
via	*vianame*	*layername*	*layername* [*float*]
width	{*layername* | *vianame*}	*float*	
spacing	{*layername* | *vianame*}	{*layername* | *vianame*}	*float*
overhang	{*layername* | *vianame*}	{*layername* | *vianame*}	*float*

.
.
.

ENDRULES

The mandatory keywords **RULES** and **ENDRULES** delimit the set of design rules. The keyword **layer** defines a layer whose name is given by *layername*. Each layer has associated with it a parasitic resistance and capacitance given in ohms per square and farads per square micron respectively. Concluding the layer definition is the preferred routing direction to be associated with the layer. The preferred direction is used for routing estimation only, the actual routing for a given layer may occur in either direction. At least one layer must be specified in the horizontal direction and at least one must be specified in the vertical direction. Any number of layers may defined. Layer definitions must precede all other rules.

The connections between layers are defined using the keyword **via** followed by the name given to the via cell and the two layers which are to be connected. The via name can then be used in subsequent width and spacing rules. The optional floating point number specifies the aspect ratio limit of the via if rectangular vias are allowed. The detail router will optimize the via dimensions (and orientation) for the given rules while maintaining constant via area. The area is determined from the specified width of the via. The default aspect ratio limit is 1.0.

The keyword **width** allows the definition of a given layer's minimum routing width or via width in microns. Similarly, the **spacing** keyword begins the definition of the minimum distance between any defined layers or vias.

The keyword **overhang** specifies that a layer or via must overlap another layer or via by the amount specified by the floating point number.

The second part of the parameter file is devoted to program control parameters. The format for this file is similar to the .Xdefaults format:

*programName***parameter* **:** *parameterValue*

where *programName* may be one of the following:

GENR - Genrows - standard cell row configuration program
MICK - Mickey global router
MINC - Mincut clustering program
PART - Tomus program (*n*-way partitioner)
SGGR - SGGR (sea-of-gates global router)
TWMC - TimberWolfMC - macro cell placement program
TWSC - TimberWolfSC - row based placement, and global routing program

The first time TimberWolf is executed on a design, the TimberWolf system automatically copies a default template for the parameter file from the **./TimberWolf/defaults** directory into the current design directory. The file copied will be **xxx.par** when row-based cells are present and **xxx.macro.par** when only macros are present in the design. Next, the user will be asked to edit the file using vi, or the editor specified in the **EDITOR** cshell environment variable. The user should edit the default values to their appropriate values. Any parameters which are common to all designs should be entered as defaults in the parameter template files. Figure 5.1.1 shows the default xxx.par template file. Note that wildcarding is permitted by preceding the parameter with an asterisk. Each parameter will be discussed in turn below.

Sample xxx.par template file

```
# This is a default parameter file for the TimberWolf system.
#  Please change the variables below to their appropriate values.
#

RULES

layer    metal    0.05        0.1E-15    vertical
layer    poly     20.0        0.1E-15    horizontal
layer    metal2   0.05        0.1E-15    vertical
via      contact  metal       poly
via      via      metal       metal2
width    metal    4.0
spacing  metal    metal       3.0
width    poly     4.0
spacing  poly     poly        3.0

spacing  metal    poly        0.0
width    contact  2.50
width    via      3.0
# this means stacked vias - a nonzero number would disallow stacked vias
spacing  contact  via     0

# this means metal must overlap the contact
overhang metal  contact  1.0

ENDRULES

# General parameters controlling the TimberWolf system.
*vertical_wire_weight   : 1.0
*vertical_path_weight   : 1.0
*rowSep                 : 1.0
*padspacing             : abut

# Parameters controlling TimberWolfMC.
#TWMC*slow              : 2

# Parameters controlling TimberWolfSC.
TWSC*feedThruWidth      : 2 layer 1
TWSC*do.global.route    : on

# Parameters controlling genrows configuration program.
GENR*feed_percentage    :   30.0
GENR*row_to_tile_spacing:   1
```

Figure 5.1.1

5.1.1. Genrows parameters

feed_percentage	*float*
graphics.wait	{ **on** | **off** }
minimum_row_len	*integer*
numrows	*integer*
rowSep	*float*
row_to_tile_spacing	*integer*

The keyword **feed_percentage** is followed by a floating point number which specifies the amount of space to be reserved for feedthrough cells. The amount of cell width reserved will be *feed_percentage* multiplied by the total width of the row-based cells. TimberWolfSC reports the feed percentage of the current execution at the bottom of the circuitName.out file if global routing has been requested.

The **graphics.wait** keyword allows the user to control whether Genrows will enter a wait state after configuring the rows. The default is to wait for the user to enter commands.

Genrows breaks the core area into tiles. The keyword **minimum_row_len** sets a limit on the size of a valid tile, that is, any tile whose width is smaller than the **minimum_row_len** will not have rows.

In the case of designs consisting only of row-based cells, the number of cell rows may be set to the value following the keyword **numrows**. This parameter has precedence over the rowSep parameter in calculating the spacing between standard cell rows.

The required keyword **rowSep** is followed by a floating point number representing the desired amount of separation between rows. The amount of separation between rows is this number times the average height of the rows. This is, if you want the row separation equal to the average row height, then this number should be 1.0. On the other hand, if you want the row separation to be twice the height of the rows, then this number should be 2.0. Normally, a value of 1.0 is appropriate.

The optional keyword **row_to_tile_spacing** allows the user to modify the distance between the edge of tile and the beginning and end of a row.

5.1.2. MINC - Mincut parameters

max_macro	*integer*
numcell_in_macro	*integer*

The optional keyword **max_macro** controls the target number of partitions and the optional keyword **numcell_in_macro** controls the maximum number of standard cells to be placed in any standard cell cluster. **WARNING**: these parameters should not be changed from their default values.

5.1.3. MICK - Mickey parameters

There are no user programmable parameters for the Mickey global router at this time.

5.1.4. PART - Tomus parameters

fast	*integer*
random.seed	*integer*
rowSep	*float*
slow	*integer*
vertical_path_weight	*float*
vertical_wire_weight	*float*

There are no required parameters for controlling the quality of the solution and the CPU time used by Tomus. That is, by default Tomus is set up to yield what we feel approximates the best attainable solutions. However, experienced Tomus users may wish to experiment with the optional parameters for

controlling the trade-off between CPU time and solution quality. The keywords **fast** and **slow** represent the set of optional parameters.

The keyword **fast** is an optional entry in the file *circuitName.par*. The inclusion of this keyword will cause Tomus to be executed about *n* times faster, where *n* is the value of the integer following the keyword fast. The quality of the placement will tend to decrease as *n* is made larger than one (the smaller the value of *n*, the better the result). For design space exploration, a value of *n* in the range of five to ten is recommended.

The keyword **slow** is an optional entry causing Tomus to be executed about *n* times longer than the default. The value of *n* is specified by the integer following the keyword slow. In some cases, you may get a slightly better result. However, only use the slow option if CPU time is of no interest to you.

The keyword **random.seed** is useful when the output data files have been deleted and the data needs to be regenerated. The random number generator seed is printed in the *circuitName.pout* file. If the *circuitName.cel* and *circuitName.par* files are identical, a second run using the same **random.seed** value will yield the exact same output.

The required keyword **rowSep** is followed by a floating point number representing the desired amount of separation between rows. The amount of separation between rows is this number times the average height of the rows. This is, if you want the row separation equal to the average row height, then this number should be 1.0. On the other hand, if you want the row separation to be twice the height of the rows, then this number should be 2.0. Normally, a value of 1.0 is appropriate.

The keyword **vertical_path_weight** is required. The floating point number represents the cost for one unit of vertical path length, given that the cost for one unit of horizontal path length is unity. This features allows the user to specify that the capacitance (or, in some sense, the delay) per unit length is different for the vertical routing layer as opposed to the horizontal routing layer. Tomus will seek to ensure that for each path specified in the *circuitName.net* file, the horizontal path length plus vertical_path_weight times the vertical path length is above the lower bound and below the upper bound for that path.

5.1.5. SGGR - Sea-of-gates global router parameters

global_routing_iterations	*integer*
min_peak_density	{ **on** | **off** }
min_total_density	{ **on** | **off** }
TWSC*SGGR	{ **on** | **off** }

If the TimberWolfSC keyword **SGGR** occurs in the parameter file, the sea-of-gates global router will be executed automatically after the completion of TimberWolfSC if the **do.global.route** keyword is present in the circuitName.par file. SGGR was developed specifically for multiple-metal-layer sea-of-gates circuits and subsequently extended to handle gate arrays and standard cell circuits. If the number of implicit feed through cells (or, implicit feeds) is not enough, SGGR will terminate with a message indicating that this may be the problem. If you get this message, you may restart TimberWolfSC (using the restart mechanism) and request the default global router. You may also leave out the SGGR keyword when you run TimberWolfSC to see how many tracks would be used by the default global router. Then, copy the *circuitName.sav* file to the *circuitName.res* file and rerun TimberWolf, specifying **SGGR**, so that it creates the input files for SGGR. Then, by executing SGGR, you can see how many fewer tracks are needed. In general, the more implicit feeds (or free vertical routing tracks over the rows), the greater the improvement yielded by SGGR over the default global router. Note that **SGGR** is a *TimberWolfSC* keyword.

By default, SGGR seeks to minimize the total channel density for standard cell circuits and seeks to minimize the peak (or maximum) channel density for gate array circuits. The user may override these defaults by specifying the keyword **min_peak_density** or the keyword **min_total_density** in the .par file. As you would expect, SGGR does a better job of minimizing the maximum channel density (possibly at the expense of higher total channel density) if **min_peak_density** is specified, or if the circuit is a gate array and no override is given. Conversely, the total number of tracks will usually be lower (possibly at the expense of a higher peak density) if **min_total_density** is specified, or if the circuit is a standard cell circuit and no override is specified.

The keyword **global_routing_iterations** followed by an integer modifies the number of global routing iterations that SGGR will perform. The default is three global routing iterations.

5.1.6. TWMC - TimberWolfMC parameters

User data parameters:

chip.aspect.ratio	*float*
core	[**initially**] *integer integer integer integer*
default.tracks.per.channel	*integer*
gridOffsetX	*integer*
gridOffsetY	*integer*
gridX	*integer*
gridY	*integer*
minimum_pad_space	*integer*
origin	*integer integer*
vertical_path_weight	*float*
vertical_wire_weight	*float*

Program control:

cost_only	{ **on** | **off** }
contiguous_pad_groups	{ **on** | **off** }
do.channel.graph	{ **on** | **off** }
do.compaction	*integer*
do.global.route	{ **on** | **off** }
fast	*integer*
graphics.wait	{ **on** | **off** }
no.graphics	{ **on** | **off** }
no.graphics.update	{ **on** | **off** }
padspacing	{ **uniform** | **abut** | **variable** | **exact** }
print_pins	{ **on** | **off** }
random.seed	*integer*
restart	{ **on** | **off** }
slow	*integer*

The keyword **chip.aspect.ratio** is followed by a floating point number which specifies the desired aspect ratio for the chip. TimberWolfMC uses this parameter to compute the dimensions of the core area. The cell placement is influenced in such a manner as to yield an aspect ratio close to the specified value.

The optional keyword **core** allows the user to specify the exact positions of the chip core area. The four integers following the **core** keyword specifying the dimensions are left side, bottom side, right side, and top side of the chip, respectively. If fixed cells are present in the *circuitName.cel* file, the chip core dimensions should be specified so that a frame of reference is available for determining the fixed cells positions. If the keyword **initially** is specified, TimberWolfMC will determine the core area after

considering the fixed cells; otherwise, TimberWolfMC is constrained to place the cells in the given core area. Unless the user is certain of the routing area of all of the cells, the keyword **initially** *should* be specified.

The optional keyword **default.tracks.per.channel** tells the compaction program and global router to allocate an additional number of tracks (above density) in each channel. The space allocated is the given number of tracks multiplied by the track pitch calculated for that direction value.

Four optional parameters are available for fixing the lower left hand corner of a cell to a grid or lattice, often useful for PCB applications. The parameters **gridX** and **gridY** define the grid to grid spacing in the horizontal and vertical directions, respectively, and the parameters **gridOffsetX** and **gridOffsetY** allow the grid to be shifted from the origin. All four must be specified simultaneously.

The optional keyword **minimum_pad_space** allows the user to specify a minimum space between the I/O pads.

The optional keyword **origin** allows the user to specify the origin of the core region. The two integers following the **origin** keyword specify the lower left corner of the core area.

The keyword **vertical_path_weight** is required. The floating point number represents the cost for one unit of vertical path length, given that the cost for one unit of horizontal path length is unity. This features allows the user to specify that the capacitance (or, in some sense, the delay) per unit length is different for the vertical routing layer as opposed to the horizontal routing layer. TimberWolfMC will seek to ensure that for each path specified in the *circuitName.net* file, the horizontal path length plus vertical_path_weight times the vertical path length is above the lower bound and below the upper bound for that path.

The keyword **cost_only** is an optional entry in the *circuitName.par* file allowing bypass of the simulated annealing placement algorithm. Its presence will result in TimberWolfMC reading the input file, generating an initial placement from the coordinates given in the input data, computing the initial cost, generating output files, and then terminating.

By default, members of pad groups are placed contiguously. In other words, no nonmember pads can be placed between the member pads. To change the setting to noncontiguous, the parameter value **off** must follow the keyword **contiguous_pad_groups**. In this case, nonmember pads could be placed between the pads.

If the optional keyword **do.channel.graph** is present the program will generate a channel graph for a given placement.

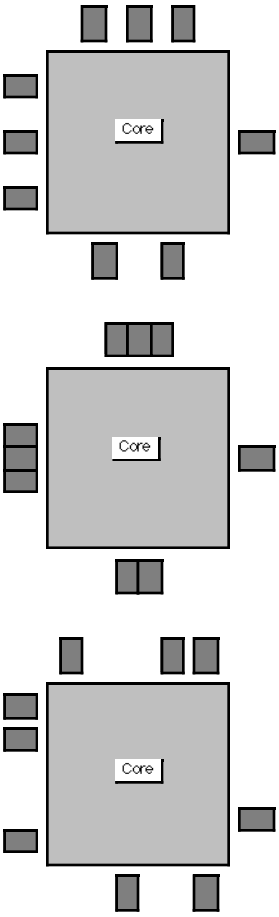
The keyword **do.global.route** causes TimberWolfMC to perform a global routing step using the channel graph generated; hence the **do.channel.graph** keyword must be present when requesting global routing.

If **do.compaction** is present in the *circuitName.par* file, the program will iterate the following flow the given number (specified by the integer) of times: compaction, channel generation, global routing. This is the placement modification phase. After each iteration, the space required for routing is accounted for by the compactor, and in the next cycle the program attempts to minimize the chip area using the current knowledge of the routing. Usually three iterations are sufficient for the placement to converge. If compaction is desired, the **do.channel.graph** and **do.global.route** keywords must be present.

By default, TimberWolfMC performs the simulated annealing placement algorithm. The user may control the run time of the simulated annealing algorithm. The keyword **fast** followed by an integer number shortens the running time of the simulated annealing algorithm by the specified integer factor

(possibly at the expense of placement quality). To increase the placement quality (at the expense of running time) use the keyword **slow** followed by an integer multiplying factor. Usually the default amount is sufficient but it is recommended that the **fast** option be used on initial runs. The placement is normally the output of the simulated annealing algorithm; however, the user may specify a placement by fixing all the cells in the *circuitName.cel* file and using the keyword **cost_only** to avoid TimberWolfMC's placement algorithm.

The graphics system has three control keywords: **no.graphics** which allows the program compiled with the X11 library to run without displaying graphics, **graphics.wait** which tells TimberWolfMC to wait for the user to enter commands after each step in the process, and **no.graphics.update** which does not update the graphics until the end of the simulated annealing run. To continue execution from a graphics wait loop, the user clicks on the FILE menu and selects CONTINUE PGM. See the section on graphics for more details concerning the graphics capabilities and commands.



The three pad spacing modes

The optional keyword **padspacing** controls the pad spacing mode. The are four modes of operation: **uniform** pad spacing, **abut**ting pad spacing, **variable** pad spacing, and **exact** pad spacing as shown in the illustration. Uniform pad spacing spaces the pads evenly on each of the sides. In the abut mode, pads are forced to touch one another. The variable pad spacing mode places each pad such that the wirelength is minimized. The last mode turns off the pad spacing algorithm and the pads remain in the place specified by the user in the *circuitName.cel* file. In the first three cases, the side and sidespace constraints are observed. The default mode is uniform padspacing.

The optional keyword **print_pins** causes TimberWolfMC to output the names of all the one pin nets in the design into the *circuitName.mout* file.

The keyword **random.seed** is useful when the output data files have been deleted and the data needs to be regenerated. The random number generator seed is printed in the *circuitName.mout* file. If the *circuitName.cel*, *circuitName.par*, and *circuitName.mest* files are identical, a second run using the same **random.seed** value will yield the exact same output. The *circuitName.mest* file should not exist if the run was the first execution of TimberWolfMC.

The optional keyword **restart** must be present in order to resume an execution of TimberWolfMC. This is useful for resuming a run after a hardware crash or other termination of a run.

5.1.7. TWSC - TimberWolfSC parameters

User data parameters:

approximately_fixed_factor	*integer*
feedThruWidth	*integer* **layer** *integer*
minimum_pad_space	*integer*
rowSep	*float*
spacer_feed_from_left	*integer*
spacer_name_twfeed	{ **on** | **off** }
spacer_width	*integer*
total_row_length	*integer*
unused_feed_name_twspacer	{ **on** | **off** }
vertical_path_weight	*float*
vertical_track_on_cell_edge	{ **on** | **off** }
vertical_wire_weight	*float*

Program control:

absolute_minimum_feeds	{ **on** | **off** }
cost_only	{ **on** | **off** }
contiguous_pad_groups	{ **on** | **off** }
create_new_cel_file	{ **on** | **off** }
do.fast.global.route	{ **on** | **off** }
do.global.route	{ **on** | **off** }
fast	*integer*
no.graphics	{ **on** | **off** }
graphics.wait	{ **on** | **off** }
no.graphics.update	{ **on** | **off** }
no_explicit_feeds	{ **on** | **off** }
no_feed_at_end	{ **on** | **off** }
old.pin.format	{ **on** | **off** }
orientation_optimization	{ **on** | **off** }
output.at.density	{ **on** | **off** }
padspacing	{ **uniform** | **abut** | **variable** | **exact** }
pin_layers_given	{ **on** | **off** }
random.seed	*integer*
restart	{ **on** | **off** }
route_only_critical_nets	{ **on** | **off** }
route_padnets_outside	{ **on** | **off** }
SGGR	{ **on** | **off** }
slow	*integer*

The keyword **approximately_fixed_factor** modifies the default window for cells which have been approximately fixed. Normally, a cell is constrained to lie within a window centered at the cell's initial position and extending one row above, one row below, and plus or minus one average cell width and three standard deviations in the horizontal direction. The user may globally increase the window size for all approximately fixed cells by entering the keyword **approximately_fixed_factor**; the new extent of the window is plus or minus **approximately_fixed_factor** rows from the initial position of the cell and plus or minus **approximately_fixed_factor** times the default window size in the horizontal direction.

The keyword **feedThruWidth** followed by an integer is required if global routing is desired. The integer informs TimberWolfSC of the width of the feed-through cells that are to be inserted should such be necessary. The **layer** keyword is followed by an integer specifying routing level where this number cross-references the layer definitions found in the RULES section of the parameter file in the order that the layers

were defined. The first layer defined in the RULES section will become layer 1, the second layer will be layer 2, and so forth. If the layer is unknown or does not matter, use layer 0.

The optional keyword **minimum_pad_space** allows the user to specify a minimum space between the I/O pads.

The required keyword **rowSep** is followed by a floating point number representing the desired amount of separation between rows. The amount of separation between rows is this number times the average height of the rows. This is, if you want the row separation equal to the average row height, then this number should be 1.0. On the other hand, if you want the row separation to be twice the height of the rows, then this number should be 2.0. Normally, a value of 1.0 is appropriate.

The optional keyword **total_row_length** is followed by an integer specifying the total available row length for a gate array circuit (only). That is, this keyword should only be used when the total row length is fixed to a value larger than the total cell width.

For gate arrays, it is often the case that the left edge of a cell must begin on a particular grid. That is, cells need not be adjacent, however, their separation must be a multiple of a certain grid. For example, in the Primary benchmark circuits (1988 International Workshop on Placement and Routing, Research Triangle Park, NC), the placement grid is 20 units. To specify this grid, the user enters the optional keyword **spacer_width** followed by an integer specifying the grid upon which the left edges of the cells must snap to.

The optional keyword **spacer_feed_from_left** followed by an integer specifies the location of an implicit feed (vertical routing track) relative to the left edge of the spacer, whose width is specified by spacer_width. For example, on the Primary gate array benchmark circuits, the 20 micron spacer is specified as:

spacer_width 20
spacer_feed_from_left 0
spacer_feed_from_left 10
spacer_feed_from_left 20

That is, each 20 micron spacer has room for three vertical routing tracks, since the track pitch is 10. In the TimberWolfSC output, the spacer is output as a cell with name GATE_ARRAY_SPACER. The leftmost implicit feed has a top pin with name SPACER_FEED_TOP_1 and a bottom pin with name SPACER_FEED_BOTTOM_1. The rightmost (and last) implicit feed pins on the spacer have names SPACER_FEED_TOP_n and SPACER_FEED_BOTTOM_n, where n is the number of implicit feeds on the spacer. Any number of spacer feeds may be specified.

The optional keyword **vertical_track_on_cell_edge** also pertains to gate arrays. As in the case of the Primary benchmarks, if implicit feeds are present on both the left and right edges of each cell, then on any two adjacent cells, two feeds (one from each cell) will overlap. The keyword **vertical_track_on_cell_edge** will eliminate the possibility of overlapping feeds due to the specification of implicit feeds on both the left and right edges of the cells.

The keyword **spacer_name_twfeed** is an optional, and is only applicable if the user has selected the gate array mode. As presented earlier, the default name of the spacer cell is GATE_ARRAY_SPACER. If the **spacer_name_twfeed** keyword is entered in the *circuitName.par* file, then the name of the spacer cell will be twfeed instead. Note that the names of the pins on the spacer cell will be the same.

Although the TimberWolfSC global routing algorithm is such that few unused feed through cells are placed into the rows, the user may want to remove all unused feed through cells. It is unwise to simply remove these from the .pl1 file since that would invalidate the global routing (as stored in the .pin file). In an attempt to aid the user in identifying which feeds in the .pl1 file are unused, the selection of the **unused_feed_name_twspacer** keyword changes the name of each unused feed from its usual name twfeed (concatenated with a unique integer) to twspacer (concatenated with a unique integer).

The keyword **vertical_path_weight** is required. The floating point number represents the cost for one unit of vertical path length, given that the cost for one unit of horizontal path length is unity. This features allows the user to specify that the capacitance (or, in some sense, the delay) per unit length is different for the vertical routing layer as opposed to the horizontal routing layer. TimberWolfSC will seek to ensure that for each path specified in the *circuitName.net* file, the horizontal path length plus vertical_path_weight times the vertical path length is above the lower bound and below the upper bound for that path.

Inclusion of the **absolute_minimum_feeds** keyword implies that the global router is to insert an absolute minimum of explicit feedthrough cells. This minimum is dictated by the placement. This keyword should not be chosen routinely since it may cause the global router to use additional routing tracks. However, for gate arrays in which explicit feeds cannot be added, this keyword can be used to advantage. For example, if the global router added a few feeds, then execute a second run using this keyword. In general, significantly fewer feeds will be used, although accompanied (usually) with an increase in routing tracks.

The keyword **cost_only** is an optional entry in the file *circuitName.par*. Its presence will result in TimberWolfSC reading the input files, generating an initial placement, computing the initial cost, generating the output files, and then graceful death. Including this keyword is highly recommended on the first run on the input files. Any errors will be directed to the output file called *circuitName.out*.

By default, members of pad groups are placed contiguously. In other words, no nonmember pads can be placed between the member pads. To change the setting to noncontiguous, the parameter value **off** must follow the keyword **contiguous_pad_groups**. In this case, nonmember pads could be placed between the pads.

The optional keyword **create_new_cel_file** instructs TimberWolfSC to create a new *circuitName.ncel* at the end of the run which contains the final placement information by using the initially keyword. The choice of **fixed**, **nonfixed**, **approximately_fixed**, or **rigidly_fixed** is based on the selection in the original circuitName.cel. For example, if a cell was originally specified with the fixed keyword, then that keyword will be used in the new *circuitName.ncel* file. If no initial placement was specified for a given cell, then the keyword **nonfixed** will be used in the new .ncel file. A new *circuitName.ncel* file will overwrite any existing *circuitName.ncel file*. Move it to *circuitName.cel* and rerun TimberWolf.

There are no required parameters for controlling the quality of the solution and the CPU time used by TimberWolfSC. That is, by default TimberWolfSC is set up to yield what we feel approximates the best attainable solutions. However, experienced TimberWolfSC users may wish to experiment with the optional parameters for controlling the trade-off between CPU time and solution quality. The keywords fast and slow represent the set of optional parameters.

The keyword **fast** is an optional entry in the file *circuitName.par*. The inclusion of this keyword will cause TimberWolfSC to be executed about *n* times faster, where *n* is the value of the integer following the

keyword **fast**. The quality of the placement will tend to decrease as *n* is made larger than one (the smaller the value of *n*, the better the result). However, it is our experience that TimberWolfSC will outperform other placement algorithms even with *n* set in such a manner that the run time of TimberWolfSC matches the run time of the other (faster) algorithm. For chip-planning applications, a value of *n* in the range of five to ten is recommended.

The keyword **slow** is an optional entry causing TimberWolfSC to be executed about *n* times longer than the default. The value of *n* is specified by the integer following the keyword slow. In some cases, you may get a slightly better result. However, only use the slow option if CPU time is of no interest to you.

The presence of the optional keyword **do.global.route** will result in TimberWolfSC completing a global routing step. Following this step, pins have been selected for interconnection in particular channels so as to minimize the total number of wiring tracks required. TimberWolfSC optimizes net segment placement (which side of a row to place a net segment, when it is switchable) so as to reduce the number of wiring tracks required. In this step, the pins for each net are assigned a group number. Pins with the same group number are to be connected. More details are available in Section 6 on the presentation of the *circuitName.pin* file.

Currently, there are two global routers: an internal global router useful for standard cell and gate array circuits which can add additional feedthroughs to complete the routing, and SGGR, a sea-of-gates global router developed specifically for multiple-metal-layer sea-of-gates circuits and subsequently extended to handle gate arrays and standard cell circuits. If the number of implicit feed through cells (or, implicit feeds) is not enough, SGGR will terminate with a message indicating that this may be the problem. If you get this message, you may restart TimberWolfSC (using the restart mechanism) and request the default global router. If the keyword **SGGR** is specified, SGGR will be executed; otherwise, the default global router will be run. In both cases, the keyword **do.global.route** must be turned on.

The graphics system has three control keywords: **no.graphics** which allows the program compiled with the X11 library to run without displaying graphics, **graphics.wait** which tells TimberWolfSC to wait for the user to enter commands after each step in the process, and **no.graphics.update** which does not update the graphics until the end of the simulated annealing run. To continue execution from a graphics wait loop, the user clicks on the FILE menu and selects CONTINUE PGM. See the section on graphics for more details concerning the graphics capabilities and commands.

The new pin output format outputs a pseudopin for every connection to a macro or pad whereas the old format only outputs a pseudopin only for nets that leave the channel. If the old format is desired, use **old.pin.format** (not recommended).

The optional keyword **orientation_optimization** instructs TimberWolfSC to perform only the following steps: read the initial placement information from the *circuitName.cel* file, optimize the orientation of the cells, and execute the global router.

If the keyword **output.at.density** is present, the placement will be output according to the density determined by the global router. This should *not* be used in the case of mixed macro/standard cell circuits.

The optional keyword **padspacing** controls the pad spacing mode. The are four modes of operation: **uniform** pad spacing, **abut**ting pad spacing, **variable** pad spacing, and **exact** pad spacing. Uniform pad spacing spaces the pads evenly on each of the sides. In the abut mode, pads are forced to touch one another. The variable pad spacing mode places each pad such that the wirelength is minimized. The last mode turns off the pad spacing algorithm and the pads remain in the place specified by the user in the *circuitName.cel*

file. In the first three cases, the side and sidespace constraints are observed. The default mode is uniform padspacing.

Normally, the pin layers must be specified. If the keyword **pin_layers_given** is assigned the parameter value **off**, the pin's layer does not need to be defined, and thus, backward compatibility with older versions of TimberWolfSC is maintained. Warning: this is only true when TimberWolfSC is run as a stand alone program.

The keyword **random.seed** is useful when the output data files have been deleted and the data needs to be regenerated. The random number generator seed is printed in the *circuitName.out* file. If the *circuitName.cel* and *circuitName.par* files are identical, a second run using the same **random.seed** value will yield the exact same output.

The optional keyword **restart** must be present in order to resume an execution of TimberWolfSC. This is useful for resuming a run after a hardware crash or other termination of a run. Crash recovery will be discussed later.

Normally, the TimberWolfSC internal global router has the freedom to insert feedthroughs to connect pins in the interior of the core area to pins on the I/O pads at the perimeter of the design if the wirelength or congestion is minimized. If the keyword **route_padnets_outside** is present, the routing will instead be forced to leave the channel in which the interior pins reside, and enter the channels which surround the core region. This will minimize feedthroughs but may lengthen the wirelength and increase the congestion outside the core area.

5.2. THE FORMAT OF THE CIRCUITNAME.CEL FILE

The file *circuitName.cel* contains the descriptions of the standard cells (row-based cells), macro cells and pads, as well as the netlist. Comments may be added using C-style comments (/* */) with the restriction that comments are no longer than 2000 characters in length. The description *must* be ordered as follows: standard cells must precede the macro cells which must precede the pads. The complete BNF for the TimberWolf system is given in Appendix A. Each entry in *circuitName.cel* for row-based cells takes on the following form:

5.2.1. Standard Cell / Gate Array Cell Definition

cell *string string*
[**legal_block_classes** *n block_class_1 .... block_class_n*]
[**swap_group** *string*]
[**ECO_added_cell**]
[**nomirror**]
[**initially** { **fixed** | **nonfixed** | **approximately_fixed** | **rigidly_fixed** } *integer* **from** {**left** | **right**} **of block** *integer*]
[**orient** { **0** | **1** | **2** | **3** }]
left *integer* **right** *integer* **bottom** *integer* **top** *integer*
pin name *string* **signal** *string* **layer** *integer integer integer*
[**equiv name** *string* **layer** *integer integer integer*]
pin name *string* **signal** *string* **layer** *integer integer integer*
[**unequiv name** *string* **layer** *integer integer integer*]
[**pin_group**
pin name *string/string* **signal** *string* **layer** *integer integer integer*
[**equiv name** *string/string* **layer** *integer integer integer*]
.
.
.
pin name *string/string* **signal** *string* **layer** *integer integer integer*
[**equiv name** *string/string* **layer** *integer integer integer*]
end_pin_group]

The keyword **cell** begins the description of a cell. The string following the keyword **cell** is ignored completely by TimberWolf. It can be used for any purpose by the user. The second string following the keyword cell must be the *name* of the cell. TimberWolf outputs the placement information in terms of these cell names.

The next line in the description of a cell is optional. The **legal_block_classes** feature allows the user to constrain each cell to a set of block classes. Here, *n* is an integer specifying how many block classes you are going to list, followed by a list of integers -- each one of which is a block class as defined in the Genrows program. The block class definitions are generated by the Genrows program and output into the *circuitName.blk* file. Genrows will allow the user to associate a class number with any of the rows as shown below in the examples. The blocks listed in the *circuitName.blk* file are listed from the top but numbered from the bottom on the screen

EXAMPLE:

cell 18 latch
legal_block_classes 3 8 5 9
left …
.
.
.

This implies that the cell named *latch* must be confined to the rows whose block class is one of 8, 5 or 9. Keep in mind that any number of rows may have been specified as having a given block class.

If a cell is to be allowed in any row, then simply don't include a **legal_block_classes** line.

EXAMPLE 2:

.blk file:

row 6
row 0 11 123 37 class 1 (first row)
row 0 63 123 89 class 1
row 0 115 123 141 class 2
row 0 187 123 193 class 2
row 0 219 123 245 class 3
row 0 271 123 297 class 3 (last row)

Suppose that the description of a cell includes the following line in the .cel file:

legal_block_classes 2 1 2

This specifies that the cell is confined to rows 1 through 4. Further, suppose that the description of a cell includes the following line:

legal_block_classes 2 1 3

This specifies that the cell is confined to rows 1 or 2, or, 5 or 6. Note that a cell in the .cel file without a **legal_block_classes l**ine can go into any of rows 1 through 6. Alternatively, you could also specify a line as follows if that was what you wanted:

legal_block_classes 3 1 2 3

Another new optional feature is the gate swapping facility. Any cells which are to participate in gate swapping must have a line as follows:

swap_group *string*

If you want gate swapping to occur between two or more cells, then you must give each of the particular cells the same **swap_group.** The gate swapping facility is accomplished by exchanging groups of pins between any cells which have the same swap_group. See the section on pin groups for more details on defining the groups of pins to be swapped.

Another optional feature is the Engineering Change Order (ECO) handling capability. This feature is useful when additional standard cells have been added to the netlist, and yet you want the placement of the "old" cells to remain essentially exactly in their positions as determined by a previous run. The "old" placement information must be given in the .cel by means of the **initially** keyword (any of fixed,

nonfixed, etc. may be used since when ECO requests are present in the *circuitName.cel* file, the placement of the "old" cells will be unchanged relative to their initial specification). Any cell having the following line within its description

ECO_added_cell

will have its initial placement information completely disregarded. Instead, a quick procedure which is a variant of force-directed placement is used to place the new cell to minimize total wire length.

If you anticipate making use of this ECO capability, then it would be very useful to *always* place the keyword **create_new_cel_file** in the *circuitName.par* file. Then, if you find that cells are to be added (or deleted), simply add (or delete) these cells from the previously generated *circuitName.ncel* file. Be sure to append the line

ECO_added_cell

to the description of the added cells and then make this new *circuitName.ncel* file to be the new *circuitName.cel* file. The execution of TimberWolf will then quickly yield an updated placement, preserving the locations of the "old" cells and placing the new cells while seeking to minimize the total wire length.

This feature is particularly useful for generating a subsequent placement which is close to that obtained from a previous run. For example, suppose a great deal of effort was made to get a particular placement correct with respect to timing restrictions. However, suppose that the designer then adds or deletes cells or nets from the .cel file, or changes the number of rows. A new TimberWolf run will generate a completely different placement, one that probably has a similar total wire length and chip area, but one with a whole new set of timing problems, etc. Hence, for such a subsequent placement, one would very much like to preserve the general character of the previous placement. This can now be achieved.

The keyword **nomirror** is optional and specifies that a cell in a row cannot have its *x*-coordinates mirrored. By default, mirroring is allowed. This should come before the preplacement option.

Preplacing a cell is optional. This is accomplished with the keyword **initially** followed by one of four choices for the subsequent keyword: **fixed, nonfixed, approximately_fixed, and rigidly_fixed**. If **fixed** is specified, the cell is to remain fixed at the specified location, whereas, **nonfixed** allows the cells to move from their specified initial positions. The integer following **fixed** or **nonfixed** represents how far from the left or right end of a row the cell should be placed initially. Following the keyword **from**, the user selects either **left** or **right** to indicate whether the integer represents a distance from the left or right end of the row. If the user selects **left**, then the distance is the amount the left edge of the cell is to be placed from the left end of the block (row). On the other hand, if the user selects **right**, then the distance is the amount the right edge of the cell is to be placed from the right end of the block. Following the keywords **of** and **block** is an integer specifying the block (row) into which the cell is to be placed initially. This integer represents a row number, in which the rows are numbered starting from the bottom of the layout. The **approximately_fixed** keyword instructs TimberWolf to keep the placement of this cell *close* to its initial position. The cell is constrained to lie within a window centered at the cell's initial position and extending one row above, one row below, and plus or minus one average cell width and three standard deviations in the horizontal direction. The user may globally increase the window size for all approximately fixed cells by entering the keyword **approximately_fixed_factor** in the .par file:

TWSC*approximately_fixed_factor: *integer*

If this keyword is found in the .par file, the extent of the window is plus or minus **approximately_fixed_factor** rows from the initial position of the cell and plus or minus **approximately_fixed_factor** times the default window size in the horizontal direction. Gate array mode is triggered by the presence of **rigidly_fixed** cells, which in turn requires the presence of the spacer_width keyword. The use of the **rigidly_fixed** keyword means that the cell *must* end up at *exactly* that coordinate.

The user may input an initial cell orientation for row-based cells through the use of the keyword **orient** followed by an integer. Note that the value of the integer must be one of 0, 1, 2, or 3. The meaning of the orientation numbers is presented in section 5.2.2.

The line beginning with the keyword **left** is required. Each keyword **left**, **right**, **bottom**, and **top** is followed by an integer representing the distance of each edge from the exact center of the cell. If a cell has an odd width, the *x*-center is truncated to form an integer. Note that the integers following **left** and **bottom** are necessarily negative and that the integers following **right** and **top** are necessarily positive. Furthermore, the following relationships must hold:

right - | left |	= 0 or 1
right + left	= 0 or 1
top - | bottom |	= 0 or 1
top - bottom	= 0 or 1

The remainder of the description of a cell indicates the position of the pins and the signal or net names associated with each pin.

Each pin description begins with the keyword **pin** and is followed by the keyword **name** which is in turn followed by a string representing the name of the pin. Following the keyword **signal** is a string representing the name of the signal or net to which this pin is to be connected.

The **layer** keyword is followed by an integer specifying routing level. This number cross-references the layer definitions found in the RULES section of the parameter file in the order that the layers were defined. The first layer defined in the RULES section will become layer 1, the second layer will be layer 2, and so forth. If the layer is unknown or does not matter, use layer 0.

The pair of integers following the layer definition represent the *x*-*y* coordinates of the location of the pin *relative* to the center of the cell. Pins do not have to be on the boundary but they may not be outside the boundary.

The description of an equivalent (or internally connected) pin begins with the keyword **equiv** and is followed by the keyword **name** which in turn is followed by a string representing the name of this pin. Next, the layer information is specified using the layer keyword followed by the routing level. Following the layer definition are two integers specifying the location of the equivalent pin relative to the center of the cell. Note that pins described by **equiv** are assumed to be connected to the same signal or net name as the last entered pin whose description begins with the keyword **pin.**

If only one of an electrically-equivalent pair of pins can be used (for example, because the resistance of the poly line connecting them is too high), the description of the equivalent pin must begin with the keyword **unequiv.** The keyword **unequiv** is followed by the keyword **name** which in turn is followed by a string representing the name of the unequivalent pin. Next are two integers specifying the location of the unequivalent pin relative to the center of the cell. Note that pins described by **unequiv** are assumed to be connected to the same signal or net name as the last entered pin whose description begins with the keyword

pin. Following a pin description beginning with **pin**, any number of pin descriptions beginning with **equiv** may follow. However, only one **unequiv** may be specified following a pin description beginning with the keyword **pin.**

An example of a cell description in which the cell is to be preplaced and mirroring is not allowed is shown below.

cell 78 ABcell8
nomirror
initially fixed 0 *from left of block* 3
left -50 *right* 50 *bottom* -25 *top* 25
pin name pin2 signal A layer 1 0 25
equiv name pin2 layer 1 0 -25

This is an example of the description of a nonpreplaced cell in which mirroring is allowed.

cell 79 ABcell11
left -50 *right* 50 *bottom* -25 *top* 25
pin name pin5 signal C layer 0 0 25
equiv name pin5 layer 0 0 -25

TimberWolf supports implicit feed throughs (or built-into-the-cell feeds), common in multiple metal circuits. The user must enter the signal name **TW_PASS_THRU** for any such pins. If implicit feed-through paths are present, TimberWolf will use such a path instead of inserting a feed-through cell whenever possible. An example follows:

cell 1 cell1
left -50 *right* 50 *bottom* -25 *top* 25
pin name 1 signal TW_PASS_THRU layer 2 30 -52
equiv name 2 layer 2 30 53
pin name 3 signal TW_PASS_THRU layer 2 -30 -52
equiv name 4 layer 2 -30 53

The optional keyword **pin_group** begins the definition of a swappable gate. The set of pins on a cell which constitute a gate are delineated as follows:

```
pin_group
     pin name 1/a  signal S14  layer 2 -9 -24
          equiv name 1/a  layer 2 -9 24
     pin name 2/a  signal S13  layer 2  -15 -24
          equiv name 2/a  layer 2  -15 24
     pin name 3/a  signal S12  layer 2  -11 -24
          equiv name 3/a  layer 2  -11 24
end_pin_group
```

Note that the keyword **pin_group** functions as a "begin" in PASCAL or a "{" in C. Similarly, **end_pin_group** functions as an "end" or "}". Obviously it is assumed that each pin group within a swap_group has the same number of pins. Note that the "/" is a *required* part of the pin name for each pin in a *gate*. When swapping two gates, the pins are swapped on a one-to-one basis. It must be the case that for any pin within a gate, there is a corresponding pin in the other gate such that the pin names prior to the

"/" match. The portion of the pin name after the "/" is used to ensure uniqueness for the complete pin name.

Note further that by defining each "gate" to consist of a single pin and by giving each cell a unique **swap_group,** then TimberWolf will in effect optimize the layout taking into account the logical equivalence of pins within each cell.

Finally, keep in mind that the pin names stay local to their original cells in the output files.

EXAMPLE:
(two cells belonging to the same swap group where each has one swappable gate)

```
cell 337 U3
swap_group U3-U4
left -16  right 16  bottom -24  top 24
pin name 00#I01  signal S1088  layer 1  11 -24
     equiv name 01#I01  layer 1   11 24
pin name 00#I02  signal S1090  layer 1   13 -24
     equiv name 01#I02  layer 1   13 24
pin_group
     pin name 5/a  signal S14  layer 1   -9 -24
          equiv name 5/a  layer 1   -9 24
     pin name 6/a  signal S13  layer 1   -15 -24
          equiv name 6/a  layer 1   -15 24
     pin name 7/a  signal S12  layer 1   -11 -24
          equiv name 7/a  layer 1   -11 24
end_pin_group
pin name 00#TW_1  signal TW_PASS_THRU   layer 1  -13 -24
     equiv name 01#TW_1   layer 1  -13 24
pin name 00#TW_2  signal TW_PASS_THRU   layer 1  -7 -24
     equiv name 01#TW_2   layer 1  -7 24
pin name 00#TW_3  signal TW_PASS_THRU   layer 1  -5 -24
     equiv name 01#TW_3   layer 1  -5 24
pin name 00#TW_4  signal TW_PASS_THRU   layer 1  -3 -24
     equiv name 01#TW_4   layer 1  -3 24
pin name 00#TW_5  signal TW_PASS_THRU   layer 1  -1 -24
     equiv name 01#TW_5 layer 1  -1 24
pin name 00#TW_6  signal TW_PASS_THRU   layer 1  1 -24
     equiv name 01#TW_6   layer 1  1 24
pin name 00#TW_7  signal TW_PASS_THRU   layer 1  3 -24
     equiv name 01#TW_7   layer 1  3 24
```

cell 448 U4
swap_group U3-U4
left -16 *right* 16 *bottom* -24 *top* 24
pin name 00#I01 signal S1088 layer 1 11 -24
equiv name 01#I01 layer 1 11 24
pin name 00#I02 signal S1089 layer 1 13 -24
equiv name 01#I02 layer 1 13 24
pin_group
pin name 5/b signal S19 layer 1 -9 -24
equiv name 5/b layer 1 -9 24
pin name 6/b signal S18 layer 1 -15 -24
equiv name 6/b layer 1 -15 24
pin name 7/b signal S17 layer 1 -11 -24
equiv name 7/b layer 1 -11 24
end_pin_group
pin name 00#TW_1 signal TW_PASS_THRU layer 1 -13 -24
equiv name 01#TW_1 layer 1 -13 24
pin name 00#TW_2 signal TW_PASS_THRU layer 1 -7 -24
equiv name 01#TW_2 layer 1 -7 24
pin name 00#TW_3 signal TW_PASS_THRU layer 1 -5 -24
equiv name 01#TW_3 layer 1 -5 24
pin name 00#TW_4 signal TW_PASS_THRU layer 1 -3 -24
equiv name 01#TW_4 layer 1 -3 24
pin name 00#TW_5 signal TW_PASS_THRU layer 1 -1 -24
equiv name 01#TW_5 layer 1 -1 24
pin name 00#TW_6 signal TW_PASS_THRU layer 1 1 -24
equiv name 01#TW_6 layer 1 1 24
pin name 00#TW_7 signal TW_PASS_THRU layer 1 3 -24
equiv name 01#TW_7 layer 1 3 24

Unused gates in a cell can also be defined. That is, the pins comprising such a gate are available to be used as pass throughs during global routing. **TW_SWAP_PASS_THRU** is the signal name for pass throughs on gates (recall that **TW_PASS_THRU** is the proper signal name for regular pass throughs on a cell).

In the example below, note that their are six gates defined among the three cells. Three of the six gates are *unused*. At the end of a TimberWolf run, any of the six gates could end up on any of the three cells.

EXAMPLE:

cell 554 U5
swap_group U5-U6-U7
left -16 *right* 16 *bottom* -24 *top* 24
pin name 00#I01 signal S1088 layer 1 11 -24
equiv name 01#I01 layer 1 11 24
pin name 00#I02 signal S1093 layer 1 13 -24
equiv name 01#I02 layer 1 13 24
pin name 00#I04 signal S22 layer 1 -9 -24
equiv name 01#I04 layer 1 -9 24
pin_group
pin name 1/a signal S21 layer 1 -15 -24
equiv name 1/a layer 1 -15 24
pin name 2/a signal S20 layer 1 -11 -24
equiv name 2/a layer 1 -11 24
end_pin_group
pin_group
pin name 1/b signal TW_SWAP_PASS_THRU layer 1 -13 -24
equiv name 1/b layer 1 -13 24
pin name 2/b signal TW_SWAP_PASS_THRU layer 1 -7 -24
equiv name 2/b layer 1 -7 24
end_pin_group
pin name 00#TW_3 signal TW_PASS_THRU layer 1 -5 -24
equiv name 01#TW_3 layer 1 -5 24
pin name 00#TW_4 signal TW_PASS_THRU layer 1 -3 -24
equiv name 01#TW_4 layer 1 -3 24
pin name 00#TW_5 signal TW_PASS_THRU layer 1 -1 -24
equiv name 01#TW_5 layer 1 -1 24
pin name 00#TW_6 signal TW_PASS_THRU layer 1 1 -24
equiv name 01#TW_6 layer 1 1 24
pin name 00#TW_7 signal TW_PASS_THRU layer 1 3 -24
equiv name 01#TW_7 layer 1 3 24

```
cell 655 U6
swap_group U5-U6-U7
left -16  right 16  bottom -24  top 24
pin name 00#I01  signal S1088   layer 1  11 -24
      equiv name 01#I01   layer 1  11 24
pin name 00#I02  signal S1097   layer 1  13 -24
      equiv name 01#I02   layer 1  13 24
pin name 00#I04  signal S25   layer 1  -9 -24
      equiv name 01#I04   layer 1  -9 24
pin_group
      pin name 1/c  signal S24   layer 1  -15 -24
            equiv name 1/c   layer 1  -15 24
      pin name 2/c  signal S23   layer 1  -11 -24
            equiv name 2/c   layer 1  -11 24
end_pin_group
pin_group
      pin name 1/d  signal TW_SWAP_PASS_THRU   layer 1  -13 -24
            equiv name 1/d  layer 1  -13 24
      pin name 2/d  signal TW_SWAP_PASS_THRU   layer 1  -7 -24
            equiv name 2/d   layer 1  -7 24
end_pin_group
pin name 00#TW_3  signal TW_PASS_THRU   layer 1  -5 -24
      equiv name 01#TW_3   layer 1  -5 24
pin name 00#TW_4  signal TW_PASS_THRU   layer 1  -3 -24
      equiv name 01#TW_4   layer 1  -3 24
pin name 00#TW_5  signal TW_PASS_THRU   layer 1  -1 -24
      equiv name 01#TW_5   layer 1  -1 24
pin name 00#TW_6  signal TW_PASS_THRU   layer 1  1 -24
      equiv name 01#TW_6   layer 1  1 24
pin name 00#TW_7  signal TW_PASS_THRU   layer 1  3 -24
      equiv name 01#TW_7   layer 1  3 24
```

```
cell 745 U7
swap_group U5-U6-U7
left -16  right 16  bottom -24  top 24
pin name 00#I01  signal S1088   layer 1  11 -24
      equiv name 01#I01   layer 1  11 24
pin name 00#I02  signal S16   layer 1  13 -24
      equiv name 01#I02   layer 1  13 24
pin name 00#I04  signal S28   layer 1  -9 -24
      equiv name 01#I04   layer 1  -9 24
pin_group
      pin name 1/e  signal S27   layer 1  -15 -24
            equiv name 1/e   layer 1  -15 24
      pin name 2/e  signal S26   layer 1  -11 -24
            equiv name 2/e   layer 1  -11 24
end_pin_group
pin_group
      pin name 1/f  signal TW_SWAP_PASS_THRU   layer 1  -13 -24
            equiv name 1/f   layer 1  -13 24
      pin name 2/f  signal TW_SWAP_PASS_THRU   layer 1  -7 -24
            equiv name 2/f   layer 1  -7 24
end_pin_group
pin name 00#TW_3  signal TW_PASS_THRU   layer 1  -5 -24
      equiv name 01#TW_3   layer 1  -5 24
pin name 00#TW_4  signal TW_PASS_THRU  layer 1   -3 -24
      equiv name 01#TW_4   layer 1  -3 24
pin name 00#TW_5  signal TW_PASS_THRU   layer 1  -1 -24
      equiv name 01#TW_5   layer 1  -1 24
pin name 00#TW_6  signal TW_PASS_THRU   layer 1  1 -24
      equiv name 01#TW_6   layer 1  1 24
pin name 00#TW_7  signal TW_PASS_THRU   layer 1  3 -24
      equiv name 01#TW_7   layer 1  3 24
```

5.2.2. Hard Macro Cell Definition

We will now describe the arbitrary rectilinear shaped cells known as macros. Hard macro cells are cells in which all geometric information is known. They are described as follows:

hardcell *string* **name** *string*
[**fixed at** *integer* **from** { **L** | **R** } *integer* **from** { **B** | **T** }]
[**fixed neighborhood**
integer **from** { **L** | **R** } *integer* **from** { **B** | **T** }
integer **from** { **L** | **R** } *integer* **from** { **B** | **T** }]
corners *integer integer integer ... integer integer*
class *integer*
orientations *integer ... integer*
pin name string **signal** string **layer** *integer integer integer*
[**equiv name** *string* **layer** *integer integer integer*]
[**instance** *string*
corners *integer integer integer ... integer integer*
class *integer*
orientations *integer ... integer*
pin name *string* **signal** *string* **layer** *integer integer integer*
equiv name *string* **layer** *integer integer integer*
]

The keyword **hardcell** begins the description of a macro cell. The string following the keyword **hardcell** indicates the cell type. It is furnished for the user's convenience but it will be ignored by TimberWolf. The string following the keyword **name** is the name of the cell. TimberWolf outputs the placement information in terms of the names of the cells.

The second construct of the hardcell format is an optional structure which specifies where a cell is to be fixed. A cell's center can be fixed at a single point or constrained to remain within a neighborhood. A point is specified by its *x* coordinate relative to either the left or right core boundary and its *y* coordinate relative to the top or bottom core edge following the keyword **fixed**. If the cell is to be constrained in a neighborhood, the user gives the two points which describe the bounding box of the neighborhood following the phrase **fixed neighborhood**. TimberWolf will not allow the center of the cell to leave the given bounding box. Since TimberWolf builds a topology which depends on cell and routing area, the user specifies the placements relative to the core edges. **L**, **R**, **B**, **T** represent the left, right, bottom, and top edges of the core, respectively. If fixed cells are present in the *circuitName.cel* file, the chip core dimensions should be specified in the circuitName.par file. See example in Figure 5.2.2.

The next line in the description of a macro cell describes the geometry of the cell and, in the process, indicates the initial placement location for the cell. TimberWolf handles cells of any rectilinear shape. Consequently, the description of the cell geometry is given in terms of a vertex list for the cell, starting from the leftmost of the lowest vertices and proceeding in a clockwise manner around the cell. The number of vertices is equal to the number of edges (or sides) of the cell (since the first vertex is not repeated). The integer *following* the keyword **corners** indicates the number of vertices (or corners) needed to describe the shape of the cell. Suppose that the value of this integer is represented by *numCorners*. TimberWolf then expects to find *numCorners* pairs of integers following the number of corners. The first pair of integers represents the *x*-*y* coordinates of the leftmost of the lowest vertices, and the last pair of integers represents the *x*-*y* coordinates of the next-leftmost of the lowest vertices. The pairs of integers need not be entered on the same line in the description file. TimberWolf generates the core area such that all points in the core area lie in the first quadrant (that is, all points have non-negative *x* and *y* values). Consequently, it is

strongly suggested that the user do the same. Since TimberWolf generates its own initial placement topology, including pad placement, the user need not be concerned with the positions of the cells. However, it is the case that the pair of integers describing the cell vertices are taken absolutely. The user is permitted to overlap the cells (or stack on top of one another).

NOTE: the order of the corners and *numCorners* has been reversed from previous versions to make the language context free. An awk script has been provided to transform from the old format to the new format To execute the script type:

reverse_corners *circuitName* <CR>

This will generate a new cell file name *circuitName.ncel*.

The next required field is the **class** keyword. It is used to specify exchange classes between cells. Only cells with the same class number may be exchanged. Normally there are no restrictions and all class fields should be set to the same non-negative number, preferably zero.

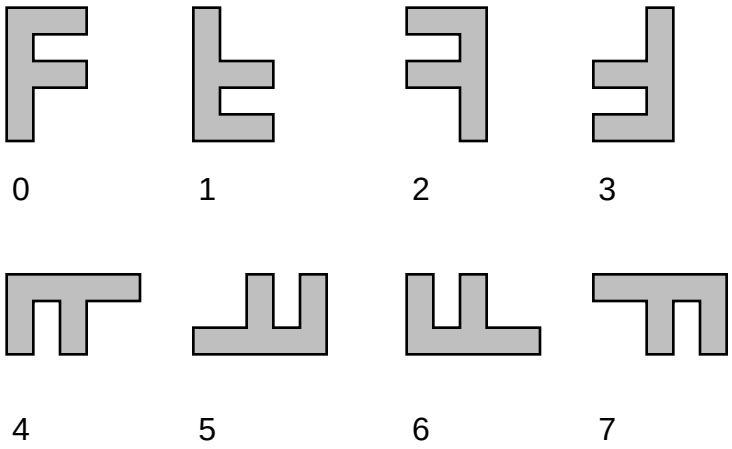


Figure 5.2.1. TimberWolf orientations.

The next required structure is the orientation list. Following the keyword **orientations** is a list of valid orientations for the cell. There are 8 possible orientations for a cell as shown in the above figure. The first integer in the list specifies the initial orientation. The cell orientation numbering scheme employed by TimberWolf is: (0) Orientation 0 is the cell geometry as described above in the vertex list. (1) Orientation 1 is the cell geometry after mirroring the *y* coordinates with respect to orientation 0. (2) Orientation 2 is the cell geometry after mirroring the *x* coordinates with respect to orientation 0. (3) Orientation 3 is the cell geometry after a rotation of 180 degrees with respect to orientation 0 (which is the same as a mirror of the *y* coordinates with respect to orientation 0 followed by a mirror of the *x* coordinates). (4) Orientation 4 is the cell geometry after a combination of a mirror of the cell's *x* coordinates followed by a 90 degree rotation of the cell with respect to orientation 0. (5) Orientation 5 is the cell geometry after a combination of a mirror of the cell's *x* coordinates followed by a -90 degree rotation of the cell with respect to orientation 0. (6) Orientation 6 is the cell geometry after a 90 degree rotation of the cell with respect to orientation 0. (7) Orientation 7 is the cell geometry after a -90 degree rotation of the cell with respect to orientation 0.

The strategy behind the numbering scheme is based on the fact that the first 4 orientations (numbered 0 through 3) have the same aspect ratio and that the second 4 orientations (numbered 4 through 7) have the same aspect ratio, an aspect ratio which is the inverse of the aspect ratio of orientations 0 through 3.

Note: The cell vertices are always input as orientation 0. If the initial orientation is nonzero, TimberWolf will perform the appropriate transformation.

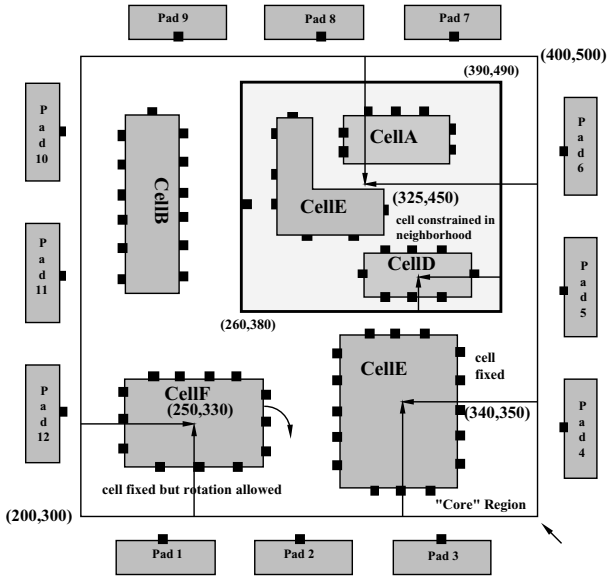


Figure 5.2.2. TimberWolf hardcell example.

This example contains a pair of rectangular cells:

hardcell 1 name cellA
corners 4 300 455 300 485 360 485 360 455
hardcell 2 name cellB
corners 4 220 390 220 485 240 485 240 390

The example also contains an L-shaped macro cell:

hardcell 3 name cellC
corners 6 270 420 270 485 290 485 290 450 330 450 330 420

In addition, cellD is constrained to be within a neighborhood:

hardcell 4 name cellD
corners 4 325 390 325 410 380 410 380 390
fixed neighborhood 140 from R 120 from T
10 from R 10 from T

We have also include two fixed cells:

hardcell 5 name cellE
corners 4 300 315 300 370 360 370 360 315
fixed at 60 from R 50 from B
class 0 orientations 0

hardcell 6 name cellF
corners 4 220 320 220 370 265 370 265 320
fixed at 50 from L 30 from B
class 0 orientations 0 1 2 3 4 5 6 7

For the circuit in Figure 5.2.2, the following line should be added in the *circuitName.par* file to specify the references properly:

*TWMC*core: initially 200 300 400 500*

TimberWolf will read the input data, set up the proper constraints, and shift origin to the default (0,0). If the user wishes for the origin to remain as shown, they should add an origin request in the *circuitName.par* file:

*TWMC*origin: 200 300*

The remainder of the description of a macro cell indicates the positions of the pins and the signal (or net) names associated with each pin. Each pin description begins with the keyword **pin** and is followed by the keyword **name**, which in turn is followed by a string representing the name of the pin. Following the keyword **signal** is a string representing the name of the signal (or net) to which this pin is to be connected. The **layer** keyword is followed by an integer specifying the routing level. This number cross-references the layer definitions found in the RULES section of the *circuitName.par* file in the order that the layers were defined. The first layer defined in the RULES section will become layer 1, the second layer will be layer 2, and so forth. If the layer is unknown or does not matter, use layer 0. The pair of integers following the layer definition represent the *absolute x-y* coordinates of the location of the pin. Pins do not have to be on the boundary but they may not be outside the boundary.

The description of an equivalent (or internally connected) pin begins with the keyword **equiv** and is followed by a string representing the name of the equivalent pin. The layer information follows as defined previously. The last two integers again represent the *absolute x-y* coordinates of the location of the equivalent pin. Note that pins described by **equiv** are assumed to be connected to the same signal or net name as the last entered pin whose description begins with the keyword **pin**. Following a pin description beginning with **pin**, any number of pin descriptions beginning with **equiv** may follow.

An example of a macro cell description, including pins, is shown below:

```
hardcell 5 name tw5
corners  8 100 100 100 800 200 800 200 400 400 400 400
         800 500 800 500 100
class 0 orientations 0 1 2 3 4 5 6 7
pin name 1 signal net1 layer 1 100 200
pin name 2 signal net2 layer 2 100 600
equiv name 3 layer 2 200 600
pin name 4 signal net3 layer 1 150 800
pin name 5 signal net4 layer 2 300 400
equiv name 6 layer 1 300 100
pin name 7 signal net5 layer 2 400 600
equiv name 8 layer 1 500 600
pin name 9 signal net6 layer 1 450 800
pin name 10 signal net7 layer 2 500 200
```

A new feature in TimberWolf is the ability to specify multiple instances of the same cell. TimberWolf will optimize the placement using the instance that will result in the best final cost. The keyword **instance** followed by a string (the instance name) begins a new instance of the current cell. The first description of the cell is known as the primary instance. All other instances are referred to by the instance name given by the user. All subsequent instances must have the same pin to signal mapping as

the primary instance. In other words, all instances must both be logically and electrically equivalent but geometric information may be changed. There is no limit to the number of cell instances. An example of a cell with two instances follows:

```
hardcell 5 name tw5
corners 8 100 100 100 800 200 800 200 400 400 400 400
          800 500 800 500 100
class 0 orientations 0 1 2 3 4 5 6 7
pin name 1 signal net1 layer 2 100 200
pin name 2 signal net2 layer 1 100 600
instance the_other_instance
corners 4  100 100 100 800 500 800 800 100
class 0 orientations 0 1 2 3 4 5 6 7
pin name 1 signal net1 layer 1 100 600
pin name 2 signal net2 layer 2 500 600
```

5.2.3. Soft Macro Cell Format

Each entry in *circuitName.cel* for soft macro cells takes on the following form:

softcell *string* **name** *string*
[**fixed at** *integer* **from** { **L** | **R** } *integer* **from** { **B** | **T** }]
[**fixed neighborhood**
    integer **from** { **L** | **R** } *integer* **from** { **B** | **T** }
    integer **from** { **L** | **R** } *integer* **from** { **B** | **T** }]
corners *integer integer integer ... integer integer*
asplb *float* **aspub** *float*
class *integer*
orientations *integer ... integer*
[**pin name** *string* **signal** string **layer** *integer integer integer*]
[**equiv name** *string* **layer** *integer integer integer*]
[**softpin name** *string* **signal** *string*]
    [**layer** *integer*]
    [**restrict side** *integer.. integer*
    [**addequiv** [**restrict side** *integer ... integer*]]
    [**equiv name** *string* **layer** *integer* [**restrict side** *integer ... integer*]]
    [**connect**]
        .
        .
        .
[**softpin name** *string* **signal** *string*]
[**pin_group** *string* { **permute** | **nopermute** }
    string { **fixed** | **nonfixed** }
        .
        .
        .
    string { **fixed** | **nonfixed** }
    [**restrict side** *integer...integer*]]

[**instance** *string*
    corners *integer integer integer ... integer integer*
    asplb *float* **aspub** *float*
    class *integer*
    orientations *integer ... integer*
    [**pin name** *string* **signal** string **layer** *integer integer integer*]
    [**equiv name** *string* **layer** *integer integer integer*]
    [**softpin name** *string* **signal** *string*]
        [**layer** *integer*]
        [**restrict side** *integer.. integer*
        [**addequiv** [**restrict side** *integer ... integer*]]
        [**equiv name** *string* **layer** *integer* [**restrict side** *integer ...integer*]]
        [**connect**]
        .
        .
        .
    [**softpin name** *string* **signal** *string*]
    [**pin_group** *string* { **permute** | **nopermute** }
        string { **fixed** | **nonfixed** }
            .
            .
            .
        string { **fixed** | **nonfixed** }
        [**restrict side** *integer...integer*]]

The keyword **softcell** begins the description of a soft macro cell. The string following the keyword **softcell** indicates the cell type. It is furnished for the user's convenience but it will be ignored by TimberWolf. The string following the keyword **name** must be the name of the cell being described.

The second construct of the cell format is an optional structure which specifies where a cell is to be fixed. A cell can be fixed at a single point or constrained to remain within a neighborhood. A point is specified by its *x* coordinate relative to either the left or right core boundary and its *y* coordinate relative to the top or bottom core edge following the keyword **fixed**. If the cell is to be constrained in a neighborhood, the user gives the two points which describe the bounding box of the neighborhood following the phrase **fixed neighborhood**. TimberWolf will not allow the center of the cell to leave the given bounding box. Since TimberWolf builds a topology which depends on cell and routing area, the user specifies the placements relative to the core edges. **L**, **R**, **B**, **T** represent the left, right, bottom, and top edges of the core, respectively.

The next line in the description of a soft cell describes the geometry of the cell. This description is given in terms of a vertex list for the cell, starting from the leftmost of the lowest vertices and proceeding in a clockwise manner around the cell. TimberWolf handles cells of any rectilinear shape. The integer *following* the keyword **corners** indicates the number of vertices (or corners) needed to describe the shape of the cell. Suppose that the value of this integer is represented by *numCorners*. TimberWolf then expects to find *numCorners* pairs of integers following the number of corners. The first pair of integers represents the *x-y* coordinates of the leftmost of the lowest vertices, and the last pair of integers represents the *x-y* coordinates of the next-leftmost of the lowest vertices. Since TimberWolf generates its own initial placement topology, including pad placement, the user need not be concerned with the positions of the cells. However, it is the case that the pair of integers describing the cell vertices are taken absolutely. The user is permitted to overlap the cells (or stack on top of one another).

NOTE: the order of the corners and *numCorners* has been reversed from previous versions to make the language context free. An awk script has been provided to transform from the old format to the new format To execute the script type:

reverse_corners *circuitName* <CR>

This will generate a new cell file name *circuitName.ncel*.

The fourth line in the description of a soft cell indicates the aspect ratio range permitted for the cell. The keyword **asplb** is followed by a floating point number which represents the lower bound on the range of permissible aspect ratios for the cell (in the orientation as implied by the previous line, which supplied the cell geometry). The keyword **aspub** is followed by a floating point number which represents the upper bound on the range of permissible aspect ratios for the cell. If the two floating point numbers have the same value, then TimberWolf assumes that the aspect ratio must remain as given.

The next required field is the **class** keyword. It is used to specify exchange classes. Only cells with the same class number may be exchanged. Normally there are no restrictions and all class fields should be set to the same non-negative number, preferably zero.

The next required structure is the orientation list. It is described in a manner analogous to the hardcell format. Again the first specified orientation is the initial orientation, and if nonzero, TimberWolf will perform the appropriate transformation.

Next, the pins of the soft cell are described. Soft cells may have any number of *hardpins* (fixed-location) and/or *softpins* (variable-location). Fixed-location pins are described in the manner presented in Section 5.2.2, that is, the descriptions begin with the keyword **pin**. Fixed pins will retain the same

relative position during aspect ratio changes, that is, fixed pins will be scaled appropriately during aspect ratio changes. Fixed-location pins may have any number of equivalent pins.

The keyword **softpin** begins a variable-location pin description, and is followed by the keyword **name** which, in turn, is followed by a string representing the name of the pin. The keyword **signal** appears next and is followed by the signal or net name associated with the pin. While the pin name and the signal information is mandatory, the user can add optional restrictions to each pin. The **layer** keyword is followed by an integer specifying routing level. This number cross-references the layer definitions found in the RULES section of the *circuitName.par* file in the order that the layers were defined. The first layer defined in the RULES section will become layer 1, the second layer will be layer 2, and so forth. If the layer is unknown or does not matter, use layer 0. In addition, restrictions may be added to the limit the edges of the cell on which a pin may appear. The keywords **restrict side** is followed by a list of integers. Each integer represents a valid cell edge for the pin where the edges must be numbered in a clockwise fashion starting from the edge represented by the first two pairs of vertices. If the **restrict side** keywords are omitted, all cell edges are valid for the softpin.

The optional keyword **addequiv** tells TimberWolf to add any number of equivalent softpins to minimize the wirelength. The user may restrict the edges of the cell that the equivalent pins may appear independent of the restrictions on the softpin itself. TimberWolf will only add equivalent pins if the wirelength is reduced and TimberWolf will name the equivalent pins.

A second method for adding equivalent pins allows the user to control the number, name, layer, and placement of the equivalent pins. The keyword **equiv** begins a softpin equivalent pin definition. The name of the equivalent pin follows the keyword **name**. The user can optionally specify the layer information of the equivalent pin using the **layer** keyword as described previously; if not supplied, TimberWolf will decide the layer. In addition, each equivalent pin may be restricted to a set of edges if desired using the **restrict side** keywords. The optional keyword **connect** tells the detail router that the equivalent pins are not preconnected and must be detail routed within the cell. A softpin may use neither, either, or both of these methods to describe its equivalent pins. The user may specify any number of equivalent pins for each softpin.

The optional **pin_group** construct allows the user to specify the common properties and ordering rules of a group of softpins. The rules are specified with the properties **fixed** and **permute**. **Permute** is a property of the pin group, and **fixed** is a property of the pin group members. If a pin group has the *permute* property, then members of this group can exist in two configurations: the given ordering or its reverse. This option is useful for placing signal buses which require that the signals be placed in ascending or descending rank. If **nopermute** is specified, then the ordering rules are decided by the members' *fixed* property. If a softpin is fixed, then its rank in the group cannot be changed; otherwise, it can be moved freely within the group.

Pin grouping is specified using the keyword **pin_group** to begin the definition of the member pins. The keyword **pin_group** is followed by a user specified group name. Pins belonging to this group *must* have unique names, and the keyword **fixed** or **nonfixed** must follow each softpin name. The pin name used in the pin group *must* be declared in the *circuitName.cel* file before the *pin_group* declaration. Each softpin can only belong to one pin group directly. The softpins belonging to the pin group must be listed in a clockwise order that follows the cell vertices, that is, they will appear bottom to top on a left cell edge, left to right on a top cell edge, top to bottom on a right cell edge, or right to left on a bottom cell edge.

Pin groups can have restrictions just like normal softpins. Furthermore, they can be used in subsequent pin groups just like ordinary softpins using the pin group name instead of a pinname in the list of softpins. As with ordinary softpins, pin groups nested within another group must have already been declared.

An example of a rectangular soft cell is show below:

```
softcell 3 name tw3
corners 4  0 0  0 400  400 400  400 0
class 0 orientations 0 1 2 3 4 5 6 7
asplb 0.80 aspub 1.20
pin name p1 signal n1 layer 1 0 100
pin name p2 signal n9 layer 2 400 100
equiv name p2_equiv layer 2 400 200
softpin name p3 signal n4
softpin name p4 signal n2 layer 1
softpin name p5 signal n5 restrict side 1 3
        addequiv
softpin name p6 signal n7
softpin name p7 signal n8
pin_group pg1 permute
p3 nonfixed
p4 nonfixed
pin_group pg2 nopermute
p6 nonfixed
pg1 nonfixed
p7 nonfixed
```

In this example, pins *p1* and *p2* are fixed location pins whereas pins *p3* thru *p7* are variable location pins. Pin *p5* is an example of a pin where the side is restricted to cell edges 1 and 3, that is, line segments (0,0) to (0,400) and (400,400) to (400,0). In addition, TimberWolf is instructed to add equivalent pins in order to minimize the wirelength of net *n5*. Two pin groups, *pg1* and *pg2*, have been specified in the example. In the first pin group, *pg1*, the two member pins *p3* and *p4* must appear consecutively (either in forward or reverse order) since the permute property has been specified. The second pin group *pg2* specifies that softpins *p6*, *p7*, and pin group *pg1* must appear consecutively but may occur in any order.

An optional feature in TimberWolf is the ability to specify multiple instances of the same cell. TimberWolf will optimize the placement using the instance that will result in the best final cost. The keyword **instance** followed by a string describing the instance name begins a new instance of the current cell. The preceding description is known as the primary instance. All other instances are referred to by the instance name given by the user. All subsequent instances must have the same pin to signal mapping as the primary instance. In other words, all instances must both be logically and electrically equivalent but geometric information may be changed. There is no limit to the number of cell instances.

5.2.4. Pad Description Format

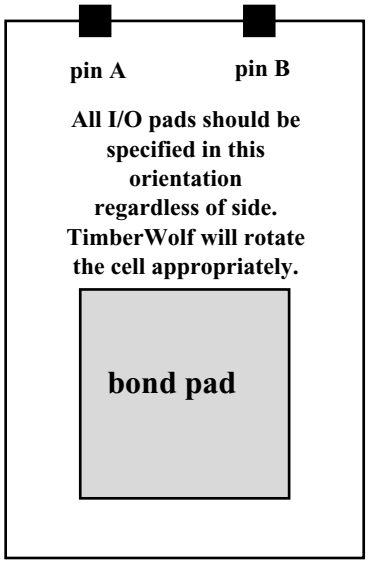
This subsection illustrates the pad description format in the input file *circuitName.cel*. Note that the description of the pads and pad groups must follow the description of all macro cells. Each entry in *circuitName.cel* must be of the following form:

pad *string* **name** *string*
corners *integer integer integer ... integer integer*
[**restrict side** { **L** , **T** , **R** , **B** , **LR**, **TB, LT, RT, LRT, etc.** }]
[**sidespace** { *float* | *float float* }]
[**pin name** *string* **signal** *string* **layer** *integer integer integer*]
[**equiv name** string **layer** *integer integer integer*]

padgroup *string* { **permute** | **nopermute** }
string { **fixed** | **nonfixed** }
.
.
.
string { **fixed** | **nonfixed** }
[**restrict side** { **L** , **T** , **R** , **B** , **LR**, **TB, LT, RT, LRT, etc.** }]
[**sidespace** { *float* | *float float* }]

The keyword **pad** begins the description of a pad. The string following the keyword pad is ignored. The string following the keyword **name** must be the name of the pad being described.

The second line in the description of a pad describes the geometry of the pad. The description of the pad geometry is given in terms of a vertex list, starting from the leftmost of the lowest vertices and proceeding in a clockwise manner around the pad. However, all pads must be given as if they were to be placed on the bottom side of the chip, that is with the bond pad at the bottom and the ports to the core at the top. TimberWolf will perform the necessary rotations in order to achieve the correct orientation for each side. Since TimberWolf places the pads relative to the core area, the absolute coordinates of the vertices are, in fact, meaningless. Hence, it is recommended that the user let the lower left corner of every pad be coincident with the origin.



The optional third line in the pad description format begins with the keyword string **restrict side** and is followed by a string containing the characters:

L , **T** , **R** , or **B**

If the restrict side option is given for the pad, TimberWolf is constrained to place the pad on the given side or sides of the chip. **L**, **T**, **R**, and **B** stand for the left, top, right, or bottom sides of the chip, respectively. More than one side may be specified. For example, **restrict side LR** allows the program to place the pad on either the left or right side of the chip where its final position is determined by the side which minimizes wirelength and satisfies pad area constraints. If no side restriction is present, TimberWolf is free to place the pad on the side which minimizes wirelength for that pad and satisfies all other constraints.

The optional keyword **sidespace** followed by a floating point number allows the user to place pads at a particular relative position. For left or right side pads (**restrict side L** or **R**), the floating point number represents a decimal fraction which specifies the fraction of the bottom-to-top span of the core of the chip to which the center of the pad is to be placed. For example, if the vertical span of the chip is 10000 microns, and if the sidespace floating point number is 0.10 for a pad to be placed on the left side, then the pad will have its *y*-center placed at 1000 microns from the bottom edge of the core. For bottom or top side pads (**restrict side B** or **T**), the floating point number represents a decimal fraction which specifies the fraction of the left-to-right span of the core to which the center of the pad is to be placed. For example, if the horizontal span of the core is 10000 microns, and if the sidespace floating point number is 0.70 for a pad to be placed on the top side, then the pad will have its *x*-center placed at 7000 microns from the left edge of the core.

Note that the **sidespace** keyword can be used to force an order on a given side. If the sidespace keyword is not present for a pad, by default, TimberWolf will place the pad so as to minimize wirelength. In summary, to constrain a pad to a side or sides of the chip, use the **restrict side** option and to constrain a pad to a particular relative position, using the **sidespace** option. The user may also use padgroups to specify an ordering. See below for details.

The optional keyword **sidespace** followed by two floating point numbers allows the user to set a valid window for pad placement. The first number represents the lower bound of the window and the second floating point number represents the upper bound. Again, each floating point number represents a decimal fraction which specifies the fraction of the left-to-right (bottom-to-top) span of the core to which the center of the pad is to be placed. It should be clear that the first form of the sidespace parameter is a compact form for the case when the lower bound equals the upper bound.

As an example, consider the following 4 pads. The first pad is unconstrained and may be placed on any side. The program will attempt to place the pad in such a way that the wirelength to the core is minimized. This is useful for subchips where the optimal I/O point is unknown. The second pad is constrained to be placed either on the top or bottom side of the core whereas the third and fourth pads must be placed on the left side of the core. In all cases, the pads are unordered on that side.

pad 11 name twpad1
corners 4 0 0 0 200 200 200 200 0
pin name p1 signal layer 1 n1 200 100

pad 12 name twpad2
corners 4 0 0 0 200 200 200 200 0
restrict side TB
pin name p2 signal layer 1 n2 200 100

pad 13 name twpad3
corners 4 0 0 0 200 200 200 200 0
restrict side L
pin name p3 signal layer 1 n3 200 100

pad 14 name twpad4
corners 4 0 0 0 200 200 200 200 0
restrict side L
pin name p3 signal layer 1 n3 200 100

The example below shows the same four pads as in the example above, however, in this case the user has requested that the pads appear at specific relative positions. Note that pads twpad3 and twpad4 are ordered bottom to top on the left side of the core. Also note that twpad2 may be placed on either the **T**, **B**, or **R** sides in a window starting at 30% and ending at 50% of the side's span.

pad 11 name twpad1
corners 4 0 0 0 200 200 200 200 0
pin name p1 signal n1 layer 1 200 100
sidespace 0.5

pad 12 name twpad2
corners 4 0 0 0 200 200 200 200 0
restrict side TBR
pin name p2 signal n2 layer 1 200 100
sidespace 0.3 0.5

pad 13 name twpad3
corners 4 0 0 0 200 200 200 200 0
restrict side L
pin name p3 signal n3 layer 1 200 100
sidespace 0.8

pad 14 name twpad4
corners 4 0 0 0 200 200 200 200 0
restrict side L
pin name p3 signal n3 layer 1 200 100
sidespace 0.9

Another way to order a group of pads is with the **padgroup** construct which allows the user to specify the common properties and ordering rules of a group of I/O pads. The rules are specified with the properties **fixed** and **permute**. **Permute** is a property of pad groups, and **fixed** is a property of the pad group members. If a pad group has the *permute* property, then members of this group can exist in two configurations: the given ordering or its reverse. This option is useful for placing I/O buses which require that the signals be placed in ascending or descending rank. If **nopermute** is specified, then the ordering

rules are decided by the members' *fixed* property. If a pad is fixed, then its rank in the group cannot be changed; otherwise, it can be moved freely within the group.

Pad grouping is specified using the keyword **padgroup** to begin the definition of the member pads. The keyword **padgroup** is followed by a user specified group name. Pads belonging to this group must have unique names, and the keyword **fixed** or **nonfixed** must follow each pad name. The pad name used in the pad group must be declared in the *circuitName.cel* file before the *padgroup* declaration. Each pad can only belong to one pad group directly. The pads belonging to the pad group must be listed in order: from left to right for **T**, **B**, and **TB** side restrictions, and bottom to top for **L**, **R**, and **LR** side restrictions.

By default, members of pad groups are placed contiguously. In other words, no nonmember pads can be placed between the member pads. To change the setting to noncontiguous, the keyword **contiguous_pad_groups** must be turned off in the *circuitName.par* file. In this case, nonmember pads will be placed between the pads.

Pad groups can have restrictions just like normal pads. Furthermore, they can be used in subsequent pads groups just like ordinary pads using the padgroupname instead of a padname in the list of pads. As with ordinary pads, pad groups nested within another group must have already been declared.

Below is an example of a pad group consisting of two ordinary pads and a previously defined padgroup which are to be placed in either increasing or decreasing order:

padgroup tw_group2 permute
twpad1 nonfixed
tw_group1 nonfixed
twpad2 nonfixed

5.3. THE FORMAT OF THE CIRCUITNAME.NET FILE

The optional file *circuitName.net* contains the critical-net timing data. The format for timing driven placement is as follows:

path *string...string* **:** *integer integer integer*

The net or list of nets following the keyword **path** describe the net or nets in this path. The nets are described using the same name (string) given in the *circuitName.cel* file. A blank space is required before and after the colon. The first integer after the colon is the lower bound constraint on the path length. If the path given is a single net, the pathlength is the wirelength of that net as measured by the half perimeter bounding box metric. The pathlength for multiple nets is the sum of the individual net wirelengths. The second integer is the upper bound on the pathlength. The third integer is the priority of the net and currently takes on the values 0 and 1. If the value of priority is 1, then the path is in the list of critical paths and TimberWolf will to place the cells in such a way that the pathlength is greater or equal to the lower bound and less than or equal to the upper bound. If the priority is 0, then TimberWolf will not use this path in the timing constraint but will print out the pathlength in the *circuitName.pth* file (as it also will for a priority of 1).

The user may issue additional directives regarding particular nets. The following format is used:

net string [**ignore**] [**do_not_global_route**]

The keyword **net** is followed by a string which must be the name of a net specified in *circuitName.cel*. The optional keyword **ignore** is used to instruct TimberWolf to ignore this net during the placement stage. However, this net will be globally routed. Nets which connect to virtually every cell, for example, clock and power/ground lines, have no influence on placement quality. However, because they connect to almost every cell, each incremental placement change in the simulated annealing algorithm requires the reevaluation of the spans of these nets. The consequence is a dramatic increase in the run time of the algorithm. The intent of this keyword is to reduce the placement time without sacrificing any placement quality.

The optional keyword **do_not_global_route** is used to instruct TimberWolf to take this net into account during placement only, that is, let it influence the placement but do not global route the net. This is typically used for nets which will be routed by some special means.

6. TIMBERWOLF OUTPUT FILES

6.1. MESSAGE FILES

All error messages are normally directed to the screen. In addition, the other messages may also be written to one of the TimberWolf output files depending on the program that is being executed. The following programs have output files:

Mickey	*circuitName.gout*
SGGR	*circuitName.sgout*
TimberWolfMC	*circuitName.mout*
TimberWolfSC	*circuitName.out*
Tomus	*circuitName.pout*

Each file has numerous statistical data concerning the executed program. Statistical data are also printed in these files following each iteration of the simulated annealing programs, TimberWolfMC, TimberWolfSC, and Tomus. Much, although not all, of the information available in these files is self explanatory. The user should become familiar with its contents.

6.2. PLACEMENT OUTPUT FILES

There are three files which contain placement information: *circuitName.mdat*, *circuitName.pl1, and circuitName.pl2*. The first file, *circuitName.mdat*, will exist if hard or soft macro cells exist in the input file *circuitName.cel*. This file has the exact same format as the *circuitName.cel* file except all hard and soft macro cells are described in their final positions. In addition, all final softpin and equivalent pin locations are output. Notice that only the chosen instance (if more than one instance exists) will be output. For convenience, the current orientation of the macro is described by the integer following the keyword **orient.** It is important to realize that the vertex list output has already been translated and/or rotated according to the current orientation.

If row-based cells exist, TimberWolf will create the *circuitName.pl1* and *circuitName.pl2* files. The file *circuitName.pl1* contains the placement information for each cell, pad, and bounding box of any macro block. Meanwhile, the file *circuitName.pl2* contains the placement information for each row, and for the pads and macro blocks. Note that *circuitName.pl2* does not contain cell placement information. The intention is this file is to aid the user in determining the proper configuration of the circuit, that is, the number of rows, the macro block placement, and the pad placement.

The format for each entry in these two files is the same, and is as shown below:

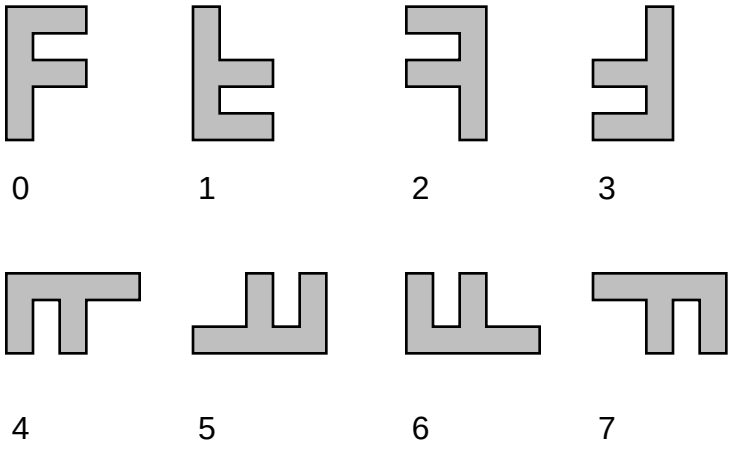
string integer integer integer integer integer integer

For *circuitName.pl1*, the string is the name of a cell, macro block, or pad. For *circuitName.pl2*, the string is the name of a macro block or pad, or the row number.

The pair of integers following the string represent the *x,y*–coordinates of the lower left corner of the cell, macro block, pad, or row. The third and fourth integers represent the *x,y*-coordinates of the upper right corner of the cell, macro block, pad, or row.

The fifth integer following the string represents the orientation of the cell, macro block, or pad. This field is set to 0 for rows. All cells were presumed to have been entered in orientation 0 in *circuitName.cel*. The orientation number given, if different from 0, represents a change in a cell orientation. The orientation numbering scheme was presented above during the description of the contents of the file *circuitName.cel*. However, due to its immediate relevance here, it will be repeated.

The TimberWolf orientation numbering scheme is as follows: Orientation 0 is that description of the geometry appearing for cells, pads and macro blocks. The other 7 possible orientations are numbered as follows: (1) Orientation 1 represents a mirror of the cell's *y*, that is, a mirror about the *x*-axis with respect to orientation 0. (2) Orientation 2 represents a mirror of the cell's *x*-coordinates, that is, a mirror about the *y*-axis with respect to orientation 0. (3) Orientation 3 represents a 180 degree rotation of the cell's coordinates with respect to orientation 0. (4) Orientation 4 represents a combination of a mirror of the cell's *x*-coordinates followed by a 90 degree rotation of the cell with respect to orientation 0. (5) Orientation 5 represents a combination of a mirror of the cell's *x*-coordinates followed by a –90 degree rotation of the cell with respect to orientation 0. (6) Orientation 6 represents a 90 degree rotation of the cell with respect to orientation 0. (7) Orientation 7 represents a –90 degree rotation of the cell with respect to orientation 0.



The sixth integer following the string represents the row number to which a cell belongs. This field is set to zero for all macro blocks. For pads on the left side of the chip, the field is set to -1. For pads on the right, bottom, and top sides of the chip, it is set to -2, -3, and -4 respectively.

The first entries in the file *circuitName.pl1* are the cell placement information. These entries are sorted by row number. That is, the cells for row number 1 appear first, then the cells for row number 2 appear, and so on. Furthermore, the cells for each row are sorted in left to right order. Following the cell placement information, the pad and macro block bounding box placement information appears. If a macro block is an arbitrary rectilinear shaped cell, the *circuitName.mdat* file should be analyzed for the final placement; the circuitName.pl1 and circuitName.pl2 only give the bounding box of the macro.

The first entries in the file *circuitName.pl2* are the row placement information. These entries are sorted by row number. Following the row placement information, the pad and macro block placement information appears.

6.3. GLOBAL ROUTING OUTPUT FILES

The global routing information is presented in an output file named *circuitName.pin*. Each line in the *circuitName.pin* file consists of 9 fields of information concerning an *active* pin. An active pin is a pin which will participate in the minimum area global route of the circuit in the following format:

string integer string string integer integer integer integer integer

Pins which are *inactive*, that is, pins which are not used in the minimum area global route, are not included in this file.

The fields for each active pin are now presented. The first field represents the name of the net attached to the pin. The second field is an integer which represents the *group* number for the pin. The group numbers are globally unique. That is, each net has its constituent pins broken down into groups such that the pins with a common group number are to be interconnected. Two pins belonging to the same net but with different group numbers are not to be interconnected. Hence, a channel or detailed router is not to be passed the net name for a pin, but rather the group number in order to achieve the minimum area layout. For clarity, the user may wish to pass both the net name and the group number (in the fashion: netName_groupNumber) so that the label for a pin produced on a plot contains the net name. In any event, the group number must be passed to the channel or detailed router in order to achieve the routing density reported by TimberWolfSC.

The third field is a string representing the name of the cell to which the pins belongs. The fourth field is a string which represents the name of the pin. The fifth and sixth field are a pair of integers representing the *x* and *y* coordinates of the location of the pin.

The seventh field represents the channel number to which the pin belongs. Horizontal core region channels are numbered (starting at 1) from the channel below the first row. The left side I/O channel (a vertical channel) is designated as channel number –1. The right side I/O channel (also a vertical channel) is designated as channel number –2. The bottom I/O channel (horizontal channel below the first core channel) is designated as channel number -3 and the top I/O channel (horizontal channel above topmost core channel). The first core region channel and the bottom I/O channel may describe the same routing region. However, in general, the bottom I/O channel is wider than first core channel and therefore a distinction can be made between connections between pins on row-based cells and pins on I/O pads. A similar situation arises in the channel above the last row.

The eighth field is a location field. It is an integer which takes on one of four possible values (-2, -1, 1, 2). A net which must leave a horizontal channel and enter a vertical channel does so by means of a pseudo pin at either the left end or right end of a horizontal channel. A pseudo pin at the left end of a horizontal channel has this location field set to -2 and a pseudo pin at the right end of a horizontal channel has this location field set to 2. The name of a pseudo pin is: PSEUDO_PIN and it is said to lie on a cell named: PSEUDO_CELL. Each left- or right-side pad pin actually appears three times! Once as a pseudo pin for a horizontal channel, once as a pseudo pin on the *bottom* (designated by -1) of a side channel, and once as a *real* pin on the *top* (designated by 1) of the same side channel.

If a pin is located at the *top* of a channel this field is set to 1. If a pin is located at the *bottom* of a channel this field is set to –1.

Pins on the top side pads have location 1 for the channel which has a number equal to the number of rows plus one. Pins on the bottom side pads have location –1 of channel number 1.

The ninth and final field is an integer indicating the layer assigned to this pin.

In summary, the nine fields for an active pin are:

1. Name of net to which the pin belongs.
2. Group number of the pin.
3. Cell name to which the pin belongs.
4. The name of the pin.
5. The *x*-coordinate of the pin location.
6. The *y*-coordinate of the pin location.
7. The channel to which the pin belongs.
8. Whether the pin is at the top (1), bottom (–1), left end (–2), or right end (2) of the channel.
9. Layer.

7. EXECUTING TIMBERWOLF

The TimberWolf system may be executed by typing **TimberWolf** on the command line assuming that the TimberWolf bin directory is included in your command search path Again, a convenience cshell script for setting the search path may be found in the TimberWolf root directory. Just execute '**source .twrc**' in the cshell to set the environment variables **TWDIR** and **DATADIR** and to include the TimberWolf bin directory in your search path. TimberWolf has the following syntax:

TimberWolf [**-gpndw**] *circuitName* [*windowId*] [*flowDirectory*]

The first optional argument to TimberWolf is the option list which is a string of letters in the set {**gpndw**} that must begin with a hyphen. If the letter **g** occurs in the string, the TimberWolf control program twflow is set to its general mode; otherwise, it is set to the TimberWolf mode. See twflow for more details. The options **p**, **n**, and **w** pertain to TimberWolf graphics. If **p** occurs in the option string, TimberWolf enter a graphics wait state to allow the user to enter commands into twflow. By default, TimberWolf immediately begins execution of the program sequence. Again, see Twflow for the list of functions available. The option **n** switches off the graphics. The default is for TimberWolf to open a graphics window. The **w** argument tells the TimberWolf graphics system to inherit a window, namely the window specified by the X *windowId*. This option is never necessary; it is only used internally by the TimberWolf system. The next argument *circuitName* is required. It is the circuit to be processed. Following *windowId* is an optional argument which specifies the flow directory to be used by TimberWolf. The default flow directory is configurable, currently it is a link in the **./TimberWolf/bin/flow** directory. To look at the default flow directory, just type **show_flows** <CR>. The available flow directories and its default will be output to the screen. The current flows supported are:

flow.noroute

flow.part

To change the default flow type **change_flow** *flowDirectory* <CR> where *flowDirectory* is one of the supported flows. The TimberWolf command line argument flowDirectory allows a design to be run using an alternate flow without having to change the system default.

8. TUTORIAL #1 - MACRO CELL DESIGN

GOAL: PLACE AND GLOBAL ROUTE A SMALL MACRO CELL DESIGN.

1) Start X server if necessary.

2) Change directory to TimberWolf root directory.

3) Run the TimberWolf initialization script:

source .twrc <CR>

You should see something similar to:

Initializing the TimberWolf environment variables...
TWDIR has been set to /twolf6/bills/TimberWolf
DATADIR has been set to /twolf6/bills/TimberWolf/DATA
DISPLAY has been set to :0
The search path now includes /twolf6/bills/TimberWolf/bin

4) You may wish to set the EDITOR environment variable if you want to use an editor other than vi. For example, you may wish to edit files using the emacs editor. Assuming emacs is in your search path, you would type:

setenv EDITOR emacs <CR>

5) Change directory to macro cell test case by typing:

cd test/macro <CR>

6) If you list the contents of the directory you should see the following files:

ls <CR>

macro.cel macro.par.sav

7) Now type TimberWolf to see its options and the available flow directories:

TimberWolf <CR>
Incorrect syntax. Correct syntax:
TimberWolf [-gpndw] designName [windowId] [flowdirectory]
whose options are one or more of the following:
g - general mode - does not use TimberWolf system information. Default is TimberWolf mode
p - pick mode - [graphics only] wait for user upon entering the program
n - no graphics - the default is to open the display and output graphics to an Xwindow
d - prints debug info and performs extensive error checking
w - parasite mode will inherit a window. Requires a valid windowId
Available installed flow directories are:
flow->flow.noroute
flow.noroute
flow.part
The current default flow directory is denoted by the arrow.
terminated abnormally with 1 error[s] and 0 warning[s]

8) The flow directory should be set to flow.noroute. If not type:

change_flow flow.noroute <CR>

9) Now execute TimberWolf on the macro design by typing:

TimberWolf macro <CR>

10) You should see TimberWolf first create a new window. In this window, you should see graphically that the syntax program is being executed on the macro design. This program parses the input files and determines the design style, in this case the macro design style. You will next see a flow diagram for the macro design style. The first step in the flow is the shell script edit_mcfiles It will open an edit window displaying the macro.par or parameter file using the editor defined in the environment variable EDITOR. Vi is the default editor. You may browse through the file but don't change anything. When you are done looking at the file, exit the file. In vi type:

:q <CR>

or in emacs type:

<Control-x><Control-c>

The programs will be executed automatically. All steps will be performed on the screen: wire estimation, placement, compaction, and global routing. Do not interrupt the program; let it progress to completion automatically.

11) When the program terminates, execute TimberWolf again by typing:

TimberWolf macro <CR>

Notice that TimberWolf automatically traverses the flow graph and exits. This is because all output files are up to date. If we wish to rerun TimberWolf, you must update the timestamp on the macro.cel file by typing:

touch macro.cel <CR>

12) Now again rerun TimberWolf on the design as before. Notice that now TimberWolfMC is executed since its output files are out of date but edit_mcfiles is not run since the parameter file has not been touched.

13) While TimberWolfMC is running, put the mouse in the menu window and click it. This will interrupt TimberWolfMC and put it in a graphics wait state awaiting user input. The menu window has the following look:

Control	Edit	Draw	Parameters
Auto Redraw	Cell Neighborhood	Draw Bins	Change Aspect Ratio
Close Graphics	Edit Cell	Draw Border	Graphics Wait
Colors	Fix Cell	Draw Globe Areas	Cancel
Continue Pgm	Fix Cell but Rot.	Draw Labels	
Dump Graphics	Group Cells	Draw Neighborhood	
Fullview	Move Cell	Draw Nets	
Graphics Update	Cancel	Draw Pins	
Redraw		Draw Single Net	
Tell Point		Draw Wiring Est	
Translate		Cancel	
Zoom			
Cancel			

Menu Window

14) We have unrolled the menus so that you can look at all the entries at once. Let's first look at the menu items under the CONTROL heading. The menu entries under the CONTROL heading are common to all graphics programs. The first entry is AUTO REDRAW which allows one to delay the redraw of the main

drawing window. Turn AUTO REDRAW OFF. Click on any of the menu items under DRAW and you will see that the screen will *not* be redrawn. This is useful if you have a large design and you want to set all of the DRAW settings at once. Now turn AUTO REDRAW ON so that the screen will redraw after a request from the user. If you select a menu accidentally, just hit CANCEL to exit that menu. Most of the items under this heading are self explanatory. Try the other functions. If you need more information on an item, turn to the graphics interface section of TimberWolfMC.

15) Now let's look at TimberWolfMC specific functions. Most of these are found under the EDIT menu heading. The EDIT CELL menu allows you to select a cell and edit its attributes, such as center of the cell, and cell orientation through a pop-up dialog box. The MOVE CELL function allows you to pick a cell and place it anywhere in the core region. The command itself will prompt you with directions. Another useful set of functions are the fix cell operations: FIX CELL, FIX CELL BUT ROTATE, and CELL NEIGHBORHOOD. FIX CELL does just that; it doesn't allow a selected cell to move relative to the core edge. The absolute position may NOT be specified since at this time we only have an estimate of the routing space required between cells. The function FIX CELL BUT ROTATE is similar except that in addition the cell is allowed to rotate around its center. The final operation CELL NEIGHBORHOOD allows a cell to move within a bounding box or neighborhood. The user picks or enters the two points (lower left and upper right corners) of the neighborhood. During annealing the cell's center will be constrained to lie within the bounds of the neighborhood. One last function is the CHANGE ASPECT RATIO function under the PARAMETER heading which modifies the aspect ratio of the core area. Aspect ratio is height divided by width. For example, an aspect ratio of 0.5 would be short and squatty. Experiment with various constraints. Again, if you need more information on an item, turn to the graphics interface section of TimberWolfMC.

16) When done exploring, use the CONTINUE PGM function under the CONTROL menu heading to continue execution of the program. At any time, the program can be interrupted by clicking on the top menu window. Allow TimberWolfMC to finish. This completes our first tutorial.

9. TUTORIAL #2 - STANDARD CELL DESIGN

GOAL: PLACE AND GLOBAL ROUTE A SMALL STANDARD CELL DESIGN USING TIMING CONSTRAINTS.

1) Initialize TimberWolf and the Xserver if necessary. (Refer to the first tutorial for the proper procedure).

2) Change directory to the macro cell test case by typing:

cd test/stdcell <CR>

3) If you list the contents of the directory you should see the following files:

ls <CR>

stdcell.cel stdcell.par.sav stdcell.net

4) In this tutorial, we will investigate timing driven placement. The timing constraints are given in the circuitName.net file. Let's look at the format of this file. Using the editor of your choice, type either:

vi stdcell.net <CR>

or

emacs stdcell.net <CR>

In this file you will see the path constraints; in this case, one constraint per net is specified. For example:

path B7 : 50 150 0

In this case, there is only one net in the critical path, B7. This stdcell circuit implements an eight bit counter. For maximum clock frequency, we would like to equalize all the net delays due to parasitics as much as possible. To do this we set a lower and upper bound on the length of each net, 50 and 150, respectively. The zero at the end of the definition tells TimberWolf to ignore this path as a constraint; instead, it will only measure the path. We will first run the design without the timing constraints. Exit the editor and run TimberWolf:

:q <CR> (vi editor)

or

<Control-x><Control-c> (emacs editor)

then

TimberWolf stdcell <CR>

5) Again the flow will run automatically. It will open the stdcell.par file to allow you to edit the parameters for the design. Notice that the number of standard cell rows have been specified:

GENR*numrows : 3

In addition, an estimate of the additional feedthroughs needed to route the design has been furnished:

GENR*feed_percentage : 3.0

When finished browsing, exit the editor.

6) TimberWolf will flow automatically into the Mincut program which prepares the design for floorplanning. Next, TimberWolf will enter the floorplanner, TimberWolfMC. Since no macro exists in this design, the Genrows program will be called immediately. You will see the graphical output of Genrows on the screen. The core region of the design is the green area. The blue rectangles are the standard cell rows and the yellow regions are extra row areas to accommodate feedthroughs. In the message window, you will see the following message:

If you wish to modify the rows, you have 10 secs.. to click on the top menu.

7) Click on the top menu by putting the mouse pointer in the window which reads:

CONTROL EDIT MERGE DRAW PARAMETERS

8) Genrows will respond by replying:

Please reconfigure the rows.

Genrows is waiting for your response...

9) Here we will look at some of the features of the Genrows row topology program. We have unrolled the Genrows menus so that you can view all the menu entries at once.

Control	Edit	Merge	Draw	Parameters
Auto Redraw	Align Macro in X	Divide Tile Left_Right	Draw Labels	Feed Percentage
Close Graphics	Align Macro in Y	Divide_Tile_Up_Down	Draw Macros	Min. Row Length
Colors	Align Rows	Limit Merges	Draw Rows	Row Separation
Continue Pgm	Edit Macro	Merge Downward	Draw Tiles	Set Spacing
Dump Graphics	Edit Row	Merge Left	Cancel	Cancel
Fullview	Edit Tile	Merge Right		
Redraw	Keep Short Row	Merge Upward		
Tell Point	Modify Core Area	Reset Tiles		
Translate	Move Macro	Cancel		
Zoom	Numrows			
Cancel	Redo			
	Restore State			
	Save State			
	Undo			
	Cancel			

10) We will experiment with different configurations but before we do, we will save the current state. To do this, click on the EDIT heading and pick the SAVE STATE menu entry. Genrows will prompt:

Enter file name for save file:

Give a name for this state. For example,

stdcell <CR>

Any number of save states are possible; just give each state a unique name.

11) Now let's explore some off Genrow's features relevent to standard cell row generation. Suppose we want four stdcell rows. We would click in the EDIT heading. When the menu unrolls, pick the entry NUMROWS. Genrows will prompt:

Enter the number of rows:

Make sure the mouse pointer is in the TimberWolf window and type:

4 <CR>

Genrows will reconfigure the design for four rows.

12) Suppose you wish to change the separation between the stdcell rows. Click the PARAMETERS heading and pick the ROW SEPARATION entry.

Genrows will prompt:

Enter row separation [1.0 nominal]:

The row separation is defined relative to the average standard cell height; the separation between adjacent rows will be the row separation multiplied by the average standard cell height. Experiment with different row separations.

13) Next change the feed percentage by clicking on the PARAMETERS heading and selecting the entry FEED PERCENTAGE. Genrows will prompt:

Enter feed ratio in percentage [0-100]:

Try 10% additional feedthroughs by entering:

10 <CR>

Try experimenting with various feed percentages.

14) Now set the design back to its original state by clicking the RESTORE STATE command under the EDIT heading. Genrows will prompt:

Enter restore file name:

Type the state:

stdcell <CR>

You will see that Genrows returns the design to its saved configuration. Now let us leave the Genrows program by selecting the CONTINUE PGM command.

15) TimberWolf will automatically enter the standard cell placement program, TimberWolfSC. While TimberWolfSC is running, put the mouse in the menu window and click it. This will interrupt TimberWolfSC and put it in a graphics wait state awaiting user input. The menu window has the following look:

Control	Draw
Auto Redraw	Draw Blocks
Close Graphics	Draw Stdcells
Colors	Draw Labels
Continue Pgm	Draw Nets
Dump Graphics	Draw Pins
Fullview	Draw Single Net
Graphics Update	Draw Single Cell Moves
Redraw	Cancel
Tell Point	
Translate	
Zoom	
Cancel	

By default, TimberWolfSC will draw the state of the placement after each update to the cost function parameter (about 160 times in all). If the DRAW SINGLE CELL MOVES command is chosen under the DRAW heading, TimberWolfSC will show the annealing process in action. Resume the execution of the run by entering the CONTINUE PGM command. You will notice that the standard cells become different colors: red denotes a highly active *hot* cell whose position changes frequently; orange denotes a *warm* cell with a lower activity; blue represents a *cold* cell whose position has remained relatively constant; and green symbolizes a *frigid* cell which has not moved from its initial position. This mode is intended for instructional purposes. Allow TimberWolfSC to complete.

16) Now let's analyze the timing constraints by using the analyze_nets program:

analyze_nets stdcell <CR>

On the screen, you will see a histogram of the individual path lengths. Notice that the spectrum of net lengths has a wide spread. Analyze_nets graphically depicts the information present in the *circuitName.pth*

file. You may wish to use the editor to browse the **stdcell.pth file**. Again, select the CONTINUE PGM command to exit this program.

17) In building this circuit, we want to equalize the net lengths as much as possible. We will use the utility net_util to build a new *circuitName.net* file. To do this type:

net_util stdcell 50 150 1 <CR>

This will create a new file **stdcell.net** which contains path constraints for each net in the design. Each path constraint will have a lower bound of 50, and an upper bound of 150. In addition, all path constraints will be activated. You may want to open an editor on the file **stdcell.net** to see the result.

18) Now we will rerun TimberWolf on the design. If we type:

TimberWolf stdcell <CR>

TimberWolf knows you have modified the circuitName.net file and reruns TimberWolfSC. Let TimberWolf run to completion.

19) Now rerun the net analysis program:

analyze_nets stdcell <CR>

You should see that the net lengths cluster around 100. There are two exceptions around 300. These are the clock (MCLOCK) and reset (MCLEAR) lines which connect to every flip flop in the design including the I/O pads. For this reason, the constraints cannot be achieved. Nets which connect to virtually every cell, for example, clock and power/ground lines, have no influence on placement quality. However, because they connect to almost every cell, each incremental placement change in the simulated annealing algorithm requires the reevaluation of the spans of these nets. The consequence is a dramatic increase in the run time of the algorithm. Through the use of the ignore keyword in the *circuitName.net* file we can ignore the evaluation of these nets during placement. Edit the **stdcell.net** file. You will need to either delete or comment out the two paths. Below is a legal comment in the *circuitName.net* file:

/
path MCLEAR : 50 150 1
path MCLOCK : 50 150 1
**/*

In addition, you will need to add the following two lines:

net MCLEAR ignore
net MCLOCK ignore

Save the file.

20) We could rerun TimberWolf and let TimberWolf figure out which programs need to be run (in this case, TimberWolfSC). However, let's explore how to force TimberWolf to execute a single program. We will use the pick mode in TimberWolf. Type:

TimberWolf -p stdcell <CR>

TimberWolf will determine the design type, draw the proper flow graph, and then wait for the user to enter a command. Pick the PICK PGM command under the FLOW menu. TimberWolf will prompt:

Pick program by clicking any mouse button at center of object

Pick TimberWolfSC with the mouse pointer. The TimberWolfSC program object will turn red. Now select the EXECUTE PGM menu under the FLOW heading. TimberWolfSC will begin execution. Once TimberWolfSC loads, interrupt the program by clicking in the menu window. Now select GRAPHICS UPDATE OFF under the CONTROL heading. TimberWolfSC will no longer update the screen after each cost function parameter adjustment. It will, however, allow the user to interrupt the program again and

file. You may wish to use the editor to browse the **stdcell.pth file**. Again, select the CONTINUE PGM command to exit this program.

17) In building this circuit, we want to equalize the net lengths as much as possible. We will use the utility net_util to build a new *circuitName.net* file. To do this type:

net_util stdcell 50 150 1 <CR>

This will create a new file **stdcell.net** which contains path constraints for each net in the design. Each path constraint will have a lower bound of 50, and an upper bound of 150. In addition, all path constraints will be activated. You may want to open an editor on the file **stdcell.net** to see the result.

18) Now we will rerun TimberWolf on the design. If we type:

TimberWolf stdcell <CR>

TimberWolf knows you have modified the circuitName.net file and reruns TimberWolfSC. Let TimberWolf run to completion.

19) Now rerun the net analysis program:

analyze_nets stdcell <CR>

You should see that the net lengths cluster around 100. There are two exceptions around 300. These are the clock (MCLOCK) and reset (MCLEAR) lines which connect to every flip flop in the design including the I/O pads. For this reason, the constraints cannot be achieved. Nets which connect to virtually every cell, for example, clock and power/ground lines, have no influence on placement quality. However, because they connect to almost every cell, each incremental placement change in the simulated annealing algorithm requires the reevaluation of the spans of these nets. The consequence is a dramatic increase in the run time of the algorithm. Through the use of the ignore keyword in the *circuitName.net* file we can ignore the evaluation of these nets during placement. Edit the **stdcell.net** file. You will need to either delete or comment out the two paths. Below is a legal comment in the *circuitName.net* file:

/
path MCLEAR : 50 150 1
path MCLOCK : 50 150 1
**/*

In addition, you will need to add the following two lines:

net MCLEAR ignore
net MCLOCK ignore

Save the file.

20) We could rerun TimberWolf and let TimberWolf figure out which programs need to be run (in this case, TimberWolfSC). However, let's explore how to force TimberWolf to execute a single program. We will use the pick mode in TimberWolf. Type:

TimberWolf -p stdcell <CR>

TimberWolf will determine the design type, draw the proper flow graph, and then wait for the user to enter a command. Pick the PICK PGM command under the FLOW menu. TimberWolf will prompt:

Pick program by clicking any mouse button at center of object

Pick TimberWolfSC with the mouse pointer. The TimberWolfSC program object will turn red. Now select the EXECUTE PGM menu under the FLOW heading. TimberWolfSC will begin execution. Once TimberWolfSC loads, interrupt the program by clicking in the menu window. Now select GRAPHICS UPDATE OFF under the CONTROL heading. TimberWolfSC will no longer update the screen after each cost function parameter adjustment. It will, however, allow the user to interrupt the program again and

enter a graphics wait state. If the user wishes to turn the graphics off completely for the remainder of the execution, select the CLOSE GRAPHICS menu under the CONTROL heading. The graphics will be disabled until the return to the master control program, TimberWolf (Twflow). Interrupt the program once again. This time select CLOSE GRAPHICS and allow TimberWolfSC to finish.

21) When control returns to TimberWolf, exit the program by selecting the EXIT PROGRAM menu under the CONTROL heading.

22) Now return to the analysis program by typing:

analyze_nets stdcell <CR>

You should see that the net lengths now all cluster around 100. Our design now has equalized net lengths. You may wish to browse the **stdcell.pth** file for more details. This concludes the second tutorial.

10. Tutorial #3 - Mixed Macro/Standard Cell Design

GOAL: PLACE AND GLOBAL ROUTE A SMALL MIXED MACRO/STANDARD CELL DESIGN.

1) Initialize TimberWolf and the Xserver if necessary. (Refer to the first tutorial for the proper procedure).

2) Change directory to the mixed cell test case by typing:

cd test/mixed <CR>

3) If you list the contents of the directory you should see the following files:

ls <CR>

mixed.cel mixed.par.sav

4) In this tutorial, we will learn more of the capabilites of Genrows. To start, run TimberWolf:

TimberWolf mixed <CR>

5) Again, the flow will be managed automatically. TimberWolf will open the mixed.par file to allow you to edit the parameters for the design. When finished browsing, exit the file.

6) TimberWolf will flow automatically into the Mincut program which prepares the design for floorplanning. The Mincut program will partition the standard cells into a cell *cluster* the size of an average macro cell in the design. These *cluster* cells become *soft* macro cells in the floorplanning problem; that is, their aspect ratios are varied and the pin locations are optimized.

7) Next, TimberWolf will enter the floorplanner, TimberWolfMC. In order to distinguish the *cluster* macro cells from the user's macro cells, interrupt TimberWolf by clicking on the top menu. Now select the DRAW WIRING EST menu from under the DRAW heading. You will see an orange border added to every user's macro cell. This wiring estimation is based on the macro's position, and pin density. The *cluster* cells have no border since their wiring area has already been factored into their area. Allow the program to continue by selecting CONTINUE PROGRAM under the CONTROL heading.

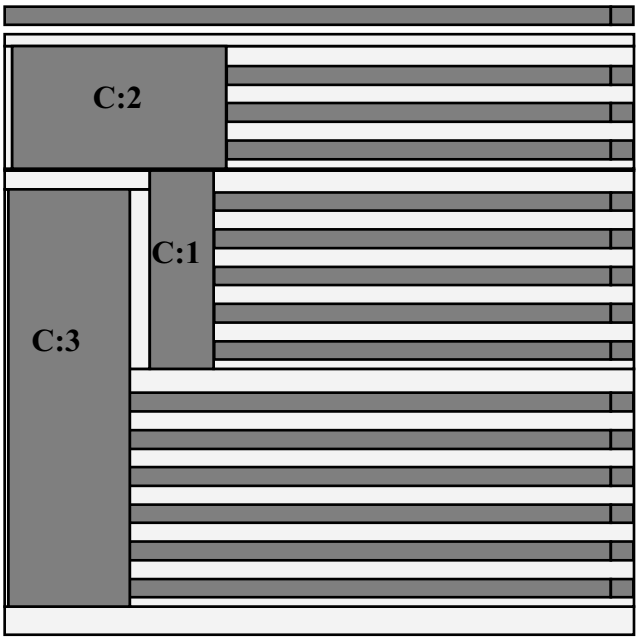


Figure 10.1

8) After floorplanning, the relative positions of the macros have been determined and TimberWolf will enter the Genrows program in order to configure the standard cell row topology. Genrows will determine a

topology based on the row separation and feed percentage, and display it on the screen. It will then wait for the user to enter a graphics command. The core region of the design is the green area. The blue rectangles are the standard cell rows, the yellow regions are extra row areas to accommodate feedthroughs, and the orange regions are macro cells. In the message window, you will see the following message:

Genrows is waiting for your response...

You should see the exact configuration shown in Figure 10.1; if not, we will use the RESTORE STATE command under the EDIT heading to return to this previously generated state. When genrows prompts:

Enter restore file name:

Type:

../mixed <CR>

Note: We have used the random seed parameter in TimberWolfMC to regenerate a previous run. Whether this state is achieved depends on the floating point arithmetic implementation of the workstation. In most cases, the RESTORE STATE command must be used.

9) For convenience, we have unrolled the Genrows menus so that you can view all the menu entries at once:

Control	Edit	Merge	Draw	Parameters
Auto Redraw	Align Macro in X	Divide Tile Left_Right	Draw Labels	Feed Percentage
Close Graphics	Align Macro in Y	Divide_Tile_Up_Down	Draw Macros	Min. Row Length
Colors	Align Rows	Limit Merges	Draw Rows	Row Separation
Continue Pgm	Edit Macro	Merge Downward	Draw Tiles	Set Spacing
Dump Graphics	Edit Row	Merge Left	Cancel	Cancel
Fullview	Edit Tile	Merge Right		
Redraw	Keep Short Row	Merge Upward		
Tell Point	Modify Core Area	Reset Tiles		
Translate	Move Macro	Cancel		
Zoom	Numrows			
Cancel	Redo			
	Restore State			
	Save State			
	Undo			
	Cancel			

10) First, turn on label drawing by selecting the DRAW LABELS menu under the DRAW heading.

11) Next, we will align two macros in the x direction. In this case, we will align the right edge of macro C:1 with the right edge of macro C:2. Click the ALIGN MACRO IN X menu under the EDIT heading. Genrows will prompt:

Select the reference macro by clicking any mouse button in cell center.

Use the mouse pointer to select cell C:2, the topmost macro. Genrows will highlight that macro and prompt:

Now pick the reference corner|center for alignment.

Select the lower right corner. A black square will appear at the selected corner. Next, Genrows will prompt:

Select the macro to be aligned by clicking any mouse button in cell center.

Use the mouse pointer to select cell C:1. At this time, C:1 will become highlighted, and the program will respond:

Now pick the corner|center you wish to align.

Use the mouse pointer to select the upper right corner of macro C:1. Genrows will align the two macros and reconfigure the rows.

12) Use ALIGN_MACRO_IN_Y under the EDIT heading to align the top of macro C:1 to the top of macro C:3. At this point the design should look like Figure 10.3:

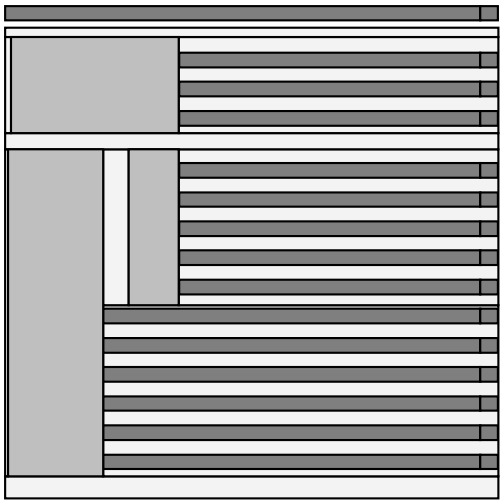


Figure 10.3 Configuration after macro alignment.

13) Next, we will experiment with the merge tile options. Momentarily, turn off the rows using the IGNORE ROWS menu under the DRAW heading. Now locate Tile:8, it is to the right of macro C:2. Now turn the rows back on using the DRAW ROWS command. Now let's perform an unlimited merge. Select MERGE DOWNWARD under the MERGE heading. Now select Tile:8. It will momentarily become highlighted before expanding downward. The row configuration should now look like Figure 10.4.

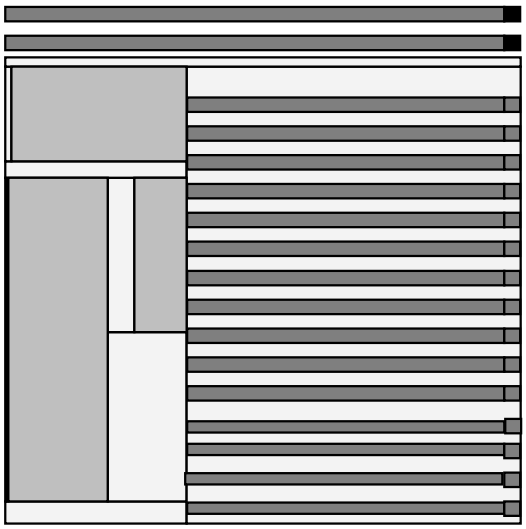
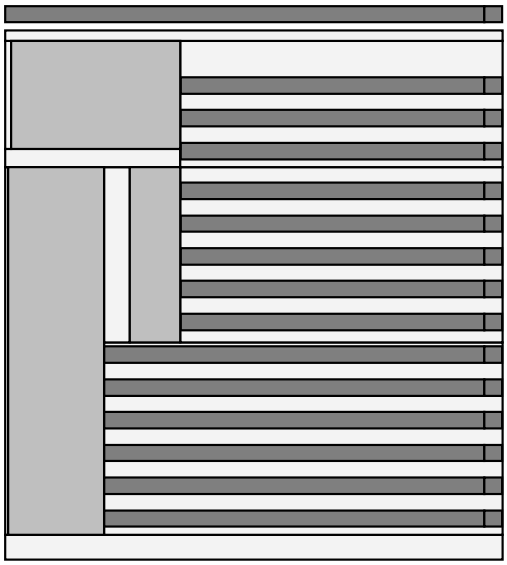


Figure 10.4 Unlimited Merge Downward

14) Now let us see how a limited merge works. Use the UNDO command to return to the previous state. Another method to set the tiles back to Genrows default configuration is to use the RESET TILES option

under the MERGE heading. Now select the LIMIT MERGES command under the MERGE heading. Again, select Tile :8 for a MERGE DOWNWARD operation. The row configuration should look like Figure 10.5. We see that the two adjacent tiles have been merged.



10.5 Limit Downward Merge

16) Perform another limited merge on Tile:8. The row configuration should look like Figure 10.6. You should now understand that the merge operations expand the size of the tile in the direction of the merge but maintain the dimension of the tile orthogonal to the merge direction.

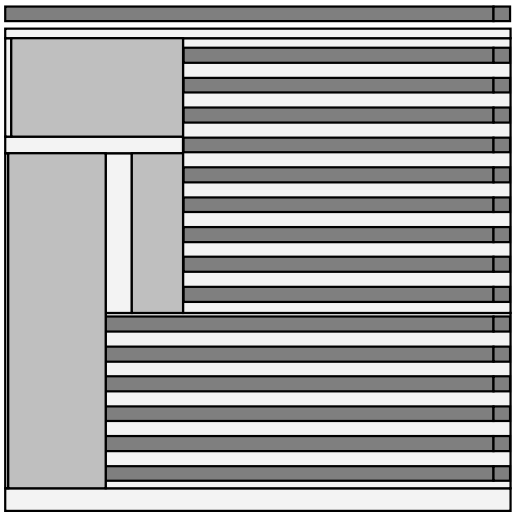


Figure 10.6. Configuration after second limited merge.

17) Next, set the row to tile edge spacing to 100 by using the SET SPACING command. Genrows will prompt for a number; enter:

100 <CR>

The rows will now be 100 units away from either the tile edge or macro edge. Notice that one or more rows are outside the core region. Click the MODIFY CORE AREA menu to increase the size of the core

region. Try to achieve the configuration shown in Figure 10.7. Save this configuration for later by using the SAVE STATE command.

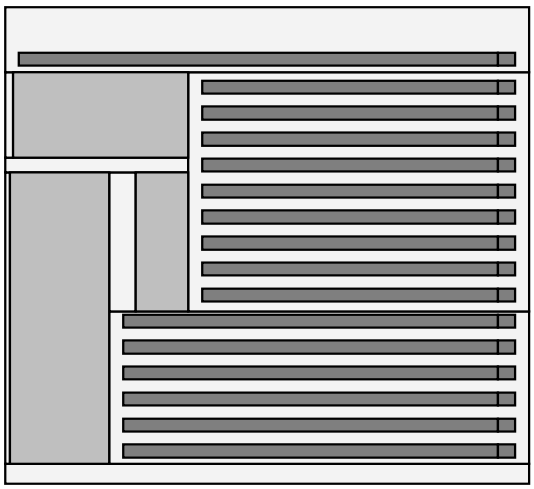


Figure 10.7 Saved configuration.

18) Now investigate the DIVIDE_TILE_LEFT_RIGHT command by selecting the large tile to the right of macro block C:2. Genrows will prompt for the dividing point. If you click in the center of the tile, you will get a configuration similar to Figure 10.8.

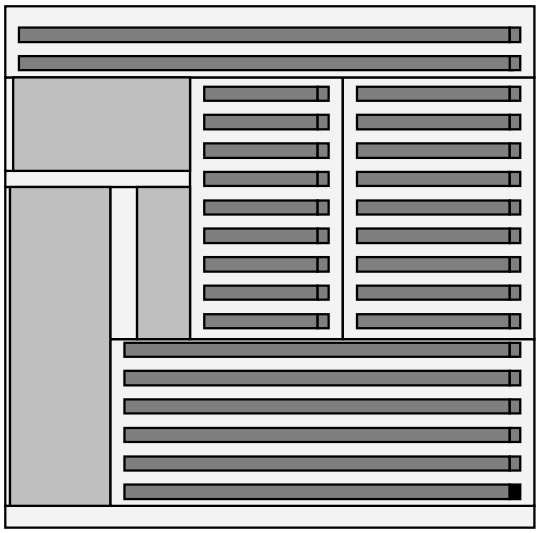


Figure 10.8 Dividing Tiles

19) Now select one of new tiles created by the division. Use EDIT TILE to change the attributes of these tiles. In addition, try the EDIT MACRO command. Refer to the graphical interface section of Genrows for more details. Experiment with various merges and divisions. When you have finished, restore the state you saved back in step 17. Exit Genrows with CONTINUE PGM.

20) Watch TimberWolfSC place the mixed design. You may stop the annealing at any time to draw the nets or change to drawing single cell moves. Allow TimberWolfSC to finish. This concludes the third tutorial.

11. TUTORIAL #4 - *N*-way Circuit Partitioning using Tomus

Goal: Divide and conquer the layout process of a mixed macro and standard cell circuit by dividing a complete chip into *n* complete sub-chips by a simulated annealing guided partitioning technique. Each of the sub-chips are placed and routed using our standard placement and routing flows described earlier. This is accomplished in parallel using *n* workstations over the network using PSC, the parallel scheduler of TimberWolfSC.

1) If you haven't invoked the X server and set the TimberWolf environment, execute steps 1-3 of Tutorial #1.

2) Change directory to the input directory with the circuit to be partitioned:

cd test/part <CR>

3) If you list the contents of the directory you should see the following files:

ls <CR>

design.cel design.par.sav

4) Now type TimberWolf to see its options and the available flow directories:

TimberWolf <CR>
Incorrect syntax. Correct syntax:
TimberWolf [-gpndw] designName [windowId] [flowdirectory]
whose options are one or more of the following:
g - general mode - does not use TimberWolf system information. Default is TimberWolf mode
p - pick mode - [graphics only] wait for user upon entering the program
n - no graphics - the default is to open the display and output graphics to an Xwindow
d - prints debug info and performs extensive error checking
w - parasite mode will inherit a window. Requires a valid windowId
Available installed flow directories are:
flow->flow.noroute
flow.noroute
flow.part
The current default flow directory is denoted by the arrow.
terminated abnormally with 1 error[s] and 0 warning[s]

5) You have two options at this point:

a) To set the default to the Tomus flow option permanently, type:

change_flow <CR>

and check out the correct syntax of the command which is

change_flow flow.part <CR>

If you chose this option, in addition you have to type the following command line to execute TimberWolf:

TimberWolf design <CR>

to start the flow.

Alternatively,

b) To set the default to the Tomus flow option just temporarily for this run of the Tomus flow, use the following command line to execute TimberWolf:

TimberWolf design flow.part <CR>

You cannot use the no-graphics option [*-n*] on the command line because Tomus is interactive graphically.

6)You should see TimberWolf first create a new window. In this window, you should see graphically that the syntax program is being executed on the input files of the design. You will next see a flow diagram for the flow using Tomus. Figure 11.1 shows the flow-chart which is called *flow.part.* The first step in the flow is the shell script edit_twfiles.

The following programs will run in sequence as described in the previous flows of TimberWolf:

1) Mincut
2) TimberWolfMC

Follow the normal instructions for the above programs as you have done before in Tutorials 1-3.

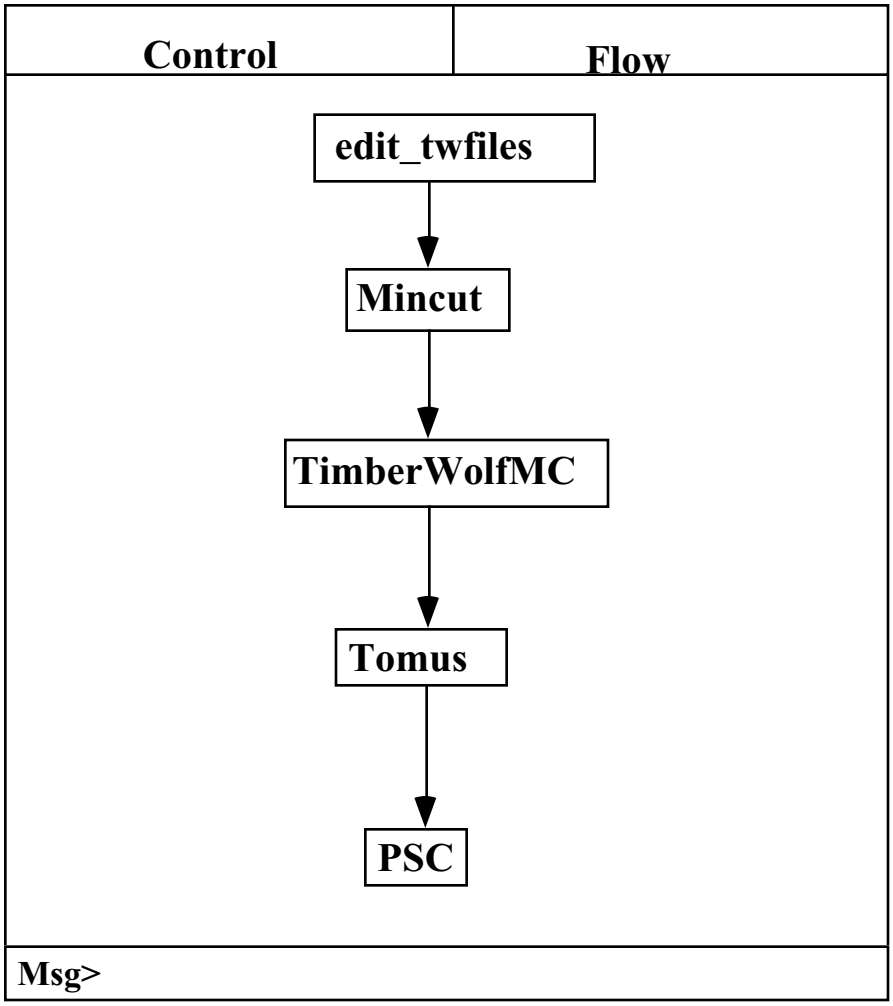


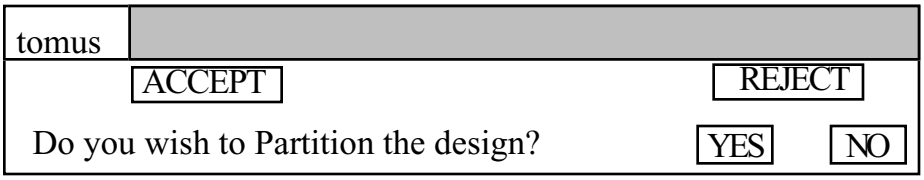
Figure 11.1

Caution: At the last stage of TimberWolfMC, the screen will show the circuit core with default rows generated by Genrows in the space available after the macros have been placed and compacted by TimberWolfMC. The purpose of this stage is twofold.

a) It gives the user an option to generate rows for a flat design, i.e. bypass Tomus and place and route flat as in the general flow without partitioning. Ignore the current rows if you want to partition the circuit.

b) It also allows the user to manually clean up any overlap which the macros might have after placement. The graphics menu of Genrows has an EDIT option which has a sub-menu MOVE MACRO which should be used at this point if necessary.

7) The first diagram you will see on the window when Tomus starts its mission is the output of TimberWolfMC. A dialog box will appear at the bottom left screen asking a question:

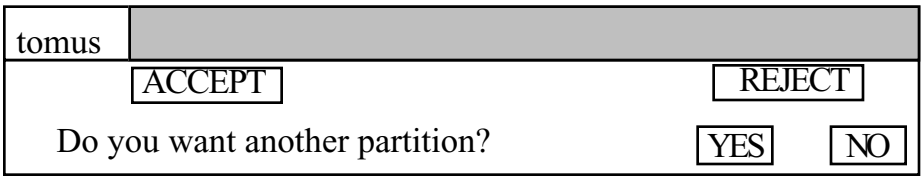


Click YES and ACCEPT unless you want to place and route flat (item 4a).

8) The window will now have the core of the circuit with the macros and the pads. The user will be prompted to define the physical locations and the size of the partitions by clicking on the core. For each *partition*, the user needs to click on the two diagonal points of the *partition*, first, the lower left corner and then the upper right corner. If the user clicks close enough to the core edges or the macro edges, Tomus will snap the partition edges to these edges. The message window at the bottom will say:

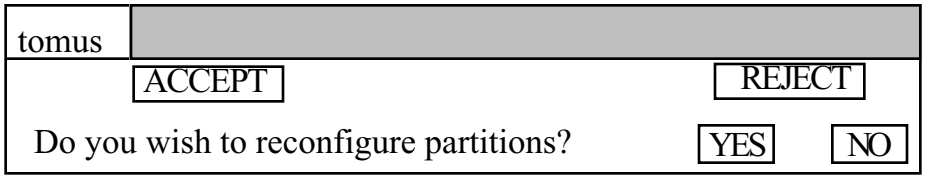
Msg > *The first point should be the lower left corner of the partition*

You should now click for the lower left corner of the first partition. The message window will then ask for the second point, the upper right corner of the partition. Note, if the partition you just created divides a macro, or an already created *partition*, you will get an error message in the message window. If the partition you created is valid, it will be painted blue to guide you for the area available for the partitions you may/may not create next. When you are done with a partition, a dialog box will appear as shown below:



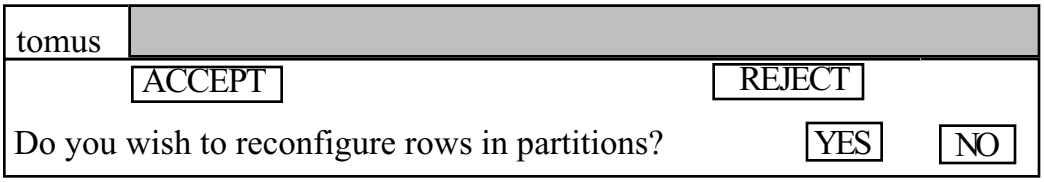
For the next partition, click on YES and repeat the above process of clicking on diagonal points of a partition. The partitions created earlier will remain painted blue. You will see the same instructions on the message window under the TimberWolf window. You will be given chances for creating as many partitions as you want through the dialog box shown above. You can terminate this step by clicking on NO. If you do not cover certain parts of the core by any partition, that area of the core won't be treated as part of the core to place standard cells.

10) At this point you will be asked a question :



If you are not satisfied with the partitions you just made, click YES and redo 9). Else continue with 11).
11) The window will draw the picture with the core as before, but now it will have additional blue lines to show the partition boundaries. It will also draw some additional cut lines in black, used internally by the program for its partitioning process. Select the DRAW menu and select sub-menu IGNORE LINES. This will allow you to see the partitions without the cut lines in them. Select the DRAW menu again and select the sub-menu DRAW NETS. This will show you the net connectivity of the circuit. Note the standard cells are now randomly clustered at the center of the tiles in each partition. To confirm this, you can select DRAWSTDCELLS from the DRAW menu and have a look at the locations of the clusters (a cluster is a set of standard cells located in a given tile) at this moment. This will show the initial state of the partitioning process.
12) Select the CONTROL menu and select sub-menu CONTINUE PROGRAM. Tomus is going to start the simulated annealing process to find you the optimal clusters of cells in the partitions you created. The message at the bottom of the window will say:
Msg > *Simulated Annealing in Progress*
13) Wait until the window redraws the state of the circuit after the annealing is completed. The message window will say :
Msg > *Partitioning done and Tomus is ready to split Partitions*
This is a pause to observe the state of the net connectivity of the circuit after partitioning and comprehend the improvement in the congestion. Select the CONTROL menu and select sub-menu CONTINUE PROGRAM when you are ready to proceed.
14) Mickey, our macro cell global router, is used to generate global routes to the nets of the circuit and pseudopads and padgroups are created based on these routes. These new pads are equivalent to I/O pads (ports) of the partitions when you visualize them as sub-chips.
The message window will say:
Msg > *Mickey in Progress; Tomus will create Pseudopads (continue)*
As Mickey completes execution, the window will redraw the circuit with the pseudopads just generated on the edges of each of the partitions. The message window will say:
Msg > *This shows pseudopads on the Partition core edges*
This is a pause to observe the pseudopads of the partitions. Select the DRAW menu and sub-menu IGNORE NETS if you don't want to view the nets again. Select the CONTROL menu and select sub-menu Continue Program when you are ready to proceed.
15) Configure rows on Partitions. The message at the bottom of the window will say
Msg > *Click in the middle of a partition to configure rows.*
Click in the middle of a partition and a new window will appear with the picture of that partition with its macro cells (if any) and some default rows configured already by Genrows. If you wish, you can modify the row parameters of the current configuration with the help of Genrows' interactive graphics. Select CONTINUE PROGRAM from the CONTROL menu and exit out of Genrows when you have finished

configuring the rows for this partition. The Tomus window with the main circuit core is going to appear again with the rows you configured in the partition just now. Continue clicking on other partitions which do not have any rows configured yet and repeat the Genrows call. You will see the same instructions on the message window under the TimberWolf window. When you are done with all the partitions, a dialog box will appear as shown below:



If you are not satisfied with the rows, click YES and redo 15). Else click NO. The message at the bottom of the window will say:

Msg > *Ready to create Inputs for TimberWolfSC.*

This is the last pause before closing the graphics window in Tomus. When you are ready, select the CONTROL menu and select sub-menu CONTINUE PROGRAM. Tomus is going to continue by generating input files for TimberWolfSC for the PSC program and terminate.

16) You will see the flow chart of the TimberWolf flow showing PSC active. PSC is going to execute next and the window will show the details of the nodes available, scheduled jobs and queued jobs. Select the PGMS menu and select EXECUTE PGMS. This will start the execution of TimberWolfSC on the nodes specified in your *nodes* file. For your convenience, we have set the *nodes* file for you for this Tutorial. Change the file when you need to add your real workstation names at your work place. TimberWolfSC graphics windows will show up on the remotes sites specified in the nodes file. Wait until the PSC window shows the status of all the jobs scheduled as *complete*. You can select the graphics menu CONTROL and sub-menu EXITPGM now. This ends the execution of the entire circuit partitioning process. Note, you can run the same Tutorial on a circuit with only standard cells.

12. GENROWS - ROW GENERATION PROGRAM

12.1. FUNCTION

The standard cell (and gate array) row topology generation is performed by the program Genrows which is spawned as either a child process of the TimberWolfMC floorplanner or the Tomus *n*-way partitioner. A valid configuration of standard cell rows must satisfy the following constraints: the standard cell rows must span the core but must not overlap the macro cells or the macro cell routing area, the total row length should be equal to the sum of the standard cell widths, and the standard cell rows should be placed at the specified row separation.

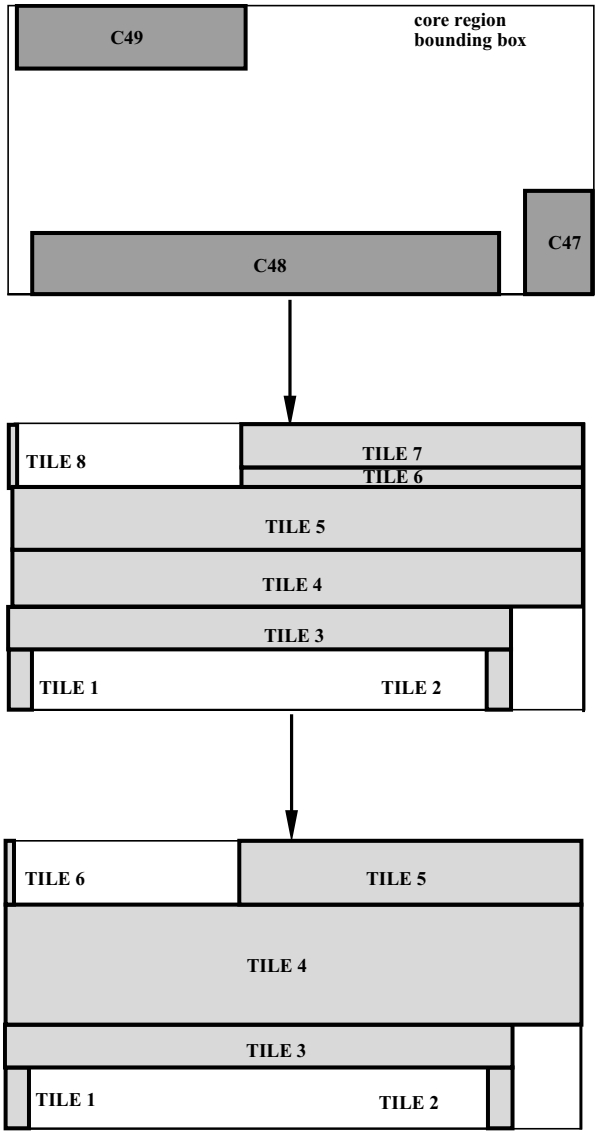


Figure 12.1.1. Tile construction and merging.

The floorplanner (or partitioner) outputs the size of the core region deemed necessary and the vertex lists of all the macro cell blocks. Any standard cell clusters, having served their purpose, are thrown away. The algorithm is as follows: First the entire core region is broken into tiles. The tiles are constructed from the spaces unoccupied by the macros and the macro cell routing. Vertically adjacent tiles are merged if their

horizontal extents match. Figure 12.1.1 shows an example of the evolution of the development of the tiles. Note that tiles 4 and 5 of the original tiles are merged as well as tiles 6 and 7.

Next, the placement of the standard cell rows is determined. Rows are first proposed for each tile. Additional rows across the top of the core are added if necessary to make the total row length exactly equal to the total cell width. Space is reserved for additional feedthrough cells. This space is controlled by the parameter file keyword **feed_percentage**; this is the additional percentage of the total row length which will be allocated for feedthroughs. The final result of Genrows is shown in Figure 12.1.2. The dark region at the end of each row is the area reserved for extra feedthrough cells. TimberWolfSC is not constrained to put the feedthroughs in this region; instead, it is constrained to place feedthroughs and row-based cells within the total row length defined by the light and dark areas.

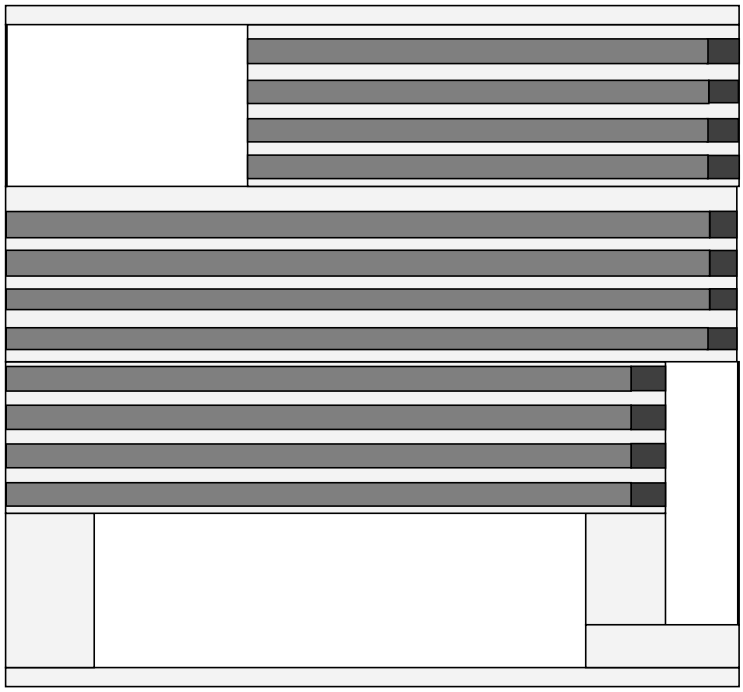


Figure 12.1.2. The final standard cell row topology.

After generation of a row topology, Genrows enters a graphics loop waiting for the user to enter commands to modify the row configuration (unless **graphics.wait** has been turned off in the *circuitName.par* file). The user may enter any command described in the graphical interface section. When satisfied, the command CONTINUE PGM is issued and Genrows will generate the files necessary for TimberWolfSC.

12.2. INPUT

The input to Genrows is the *circuitName.mver* file. It is an internal file and subject to change.

12.3. OUTPUT

The output of *Genrows* is the circuitName.blk file and its description is as follows:

rows *integer*
row *integer integer integer integer*
[**except** *integer integer* ••• **except** *integer integer*]
[**class** *integer*] [**mirror**]
row *integer integer integer integer*
[**except** *integer integer* ••• **except** *integer integer*]
[**class** *integer*] [**mirror**]
•
•
•
row *integer integer integer integer*
[**except** *integer integer* ••• **except** *integer integer*]
[**class** *integer*] [**mirror**]

The keyword **rows** is followed by an integer specifying the number of rows for the circuit. The description of each row begins with the keyword **row** followed by four integers. The first pair of these integers is the absolute *x* and *y* coordinates (respectively) of the lower left corner of the row. The second pair of these integers is the absolute *x* and *y* coordinates (respectively) of the upper right corner of the row.

TimberWolfSC handles *split* rows, which are due to the presence of macro cells on the chip. If a portion of a row is covered by a macro cell, the **except** keyword is used to indicate this. This keyword is followed by two integers specifying the range (in the *x* direction) of the row which is not available for the placement of standard cells. Note that any number of *excepts* can be specified, that is, a row can be split any number of times.

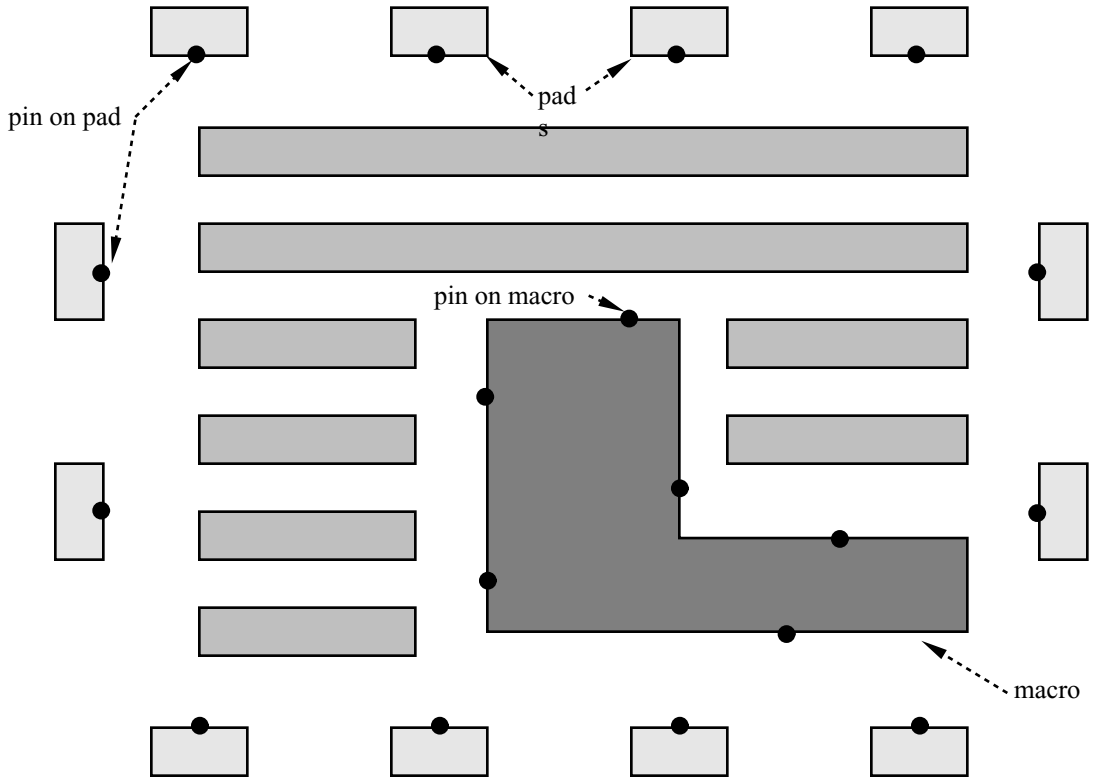


Figure 12.3.1

A hypothetical example of a mixed macro/standard cell chip is shown in Figure 12.3.1. Genrows would output the following:

```
rows 6
row   0    0    400   25     except  112  400
row   0   50    400   75     except  112  400
row   0  100    400  125     except  112  275
row   0  150    400  175     except  112  275
row   0  200    400  225
row   0  250    400  275
```

12.4. GRAPHICAL INTERFACE

Control	Edit	Merge	Draw	Parameters
Auto Redraw	Align Macro in X	Divide Tile Left_Right	Draw Labels	Feed Percentage
Close Graphics	Align Macro in Y	Divide_Tile_Up_Down	Draw Macros	Min. Row Length
Colors	Align Rows	Limit Merges	Draw Rows	Row Separation
Continue Pgm	Edit Macro	Merge Downward	Draw Tiles	Set Spacing
Dump Graphics	Edit Row	Merge Left	Cancel	Cancel
Fullview	Edit Tile	Merge Right		
Redraw	Keep Short Row	Merge Upward		
Tell Point	Modify Core Area	Reset Tiles		
Translate	Move Macro	Cancel		
Zoom	Numrows			
Cancel	Redo			
	Restore State			
	Save State			
	Undo			
	Cancel			

Control Menu

{**Auto Redraw On** | **Auto Redraw Off**} - toggles between automatic redraw when part of the screen has been covered (on) and suppresssion of this feature (off).

Close Graphics - at any time the user can close the windows and return to batch mode.

Colors - individual colors may turned off or on using a dialogbox.

Continue Prog. - breaks the graphics wait loop and continues execution of the program.

Dump Graphics - dumps the contents of the screen to files in the directory defined by the environment variable $DATADIR defined in the current cshell. The *Draw* program can be used to display this data at a later date.

Fullview - returns the user to full view of the main window.

{**Graphics Update On** | **Graphics Update Off**} - toggles between automatic redraw after each adjustment of the cost function parameters (on) and the inhibition of the redraws (off). Interrupt capability remains available in this mode.

Redraw - refresh the display.

Tell Point - returns the user coordinate under the mouse pointer when any mouse button is clicked in the main window.

Translate - performs a reorigin of the main window. The coordinate under the mouse pointer when any mouse button is clicked in the main window will become the new center of the main window.

Zoom - allows the user to zoom in or out to any part of the screen. The zoom (in) area is accomplished by either picking the lower left and upper right points of the zoom rectangle or by entering them in the message window. The points are entered as two integers separated by commas, ie. 20, 40 for the point (20,40). Note: the pointer must be in the message window to enter points through the keyboard. If the points are entered in the reverse order (upper right, lower left), a zoom out will be performed.

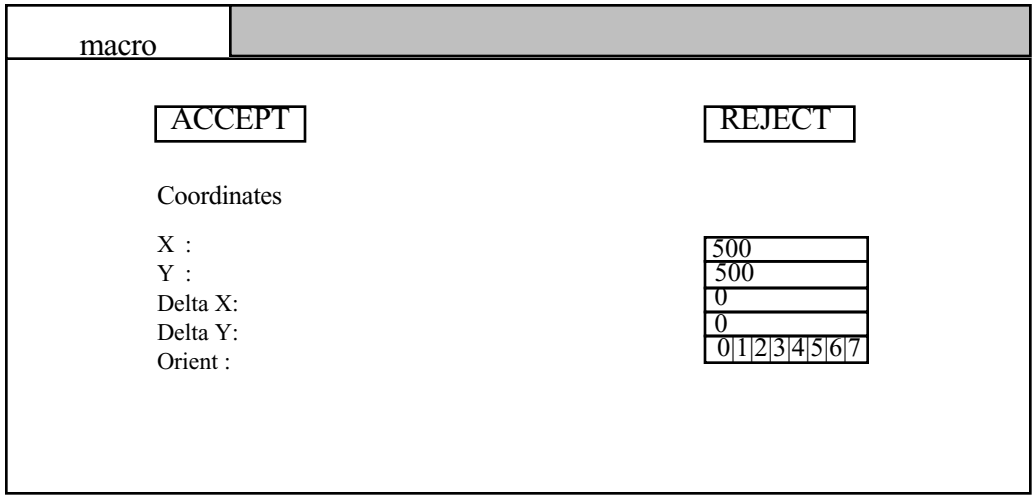
Edit Menu

Align Macro in X[1] - used to align the x-coordinates of the center or corner of one macro cell to the center or corner of another macro cell. The alignment process is as follows: First select a macro which will become the reference for alignment. The selected macro will become highlighted. Next, select the reference corner. A black square will appear on the chosen corner or center of the macro cell. Now select the macro which is to be aligned to this reference macro. Now this cell will become highlighted. Select the corner or center of the macro to be aligned. Genrows will respond by aligning the second selected macro to the reference macro. If you are not satisfied with the alignment, use the UNDO command described below.

Align Macro in Y[1] - used to align the y-coordinates of the center or corner of one macro cell to the center or corner of another macro cell. The process is similar to described in Align Macro in X.

Align Rows - used to align the rows contained in two different tiles. Genrows automatically aligns as many rows as possible; this function is a convenient way of setting the channel separation to be the same for two chosen tiles. The alignment process is as follows: Select the reference tile. It will become highlighted. Now select the tile whose rows need to be aligned. This tile's channel width will be set to the reference tiles channel separation.

Edit Macro[1] - used to edit the position and orientation of a macro. To edit a macro, first select a macro with the mouse pointer. This picked macro will become highlighted. Now pick the reference point on the macro. A dialog box will appear on the screen which looks like the following:

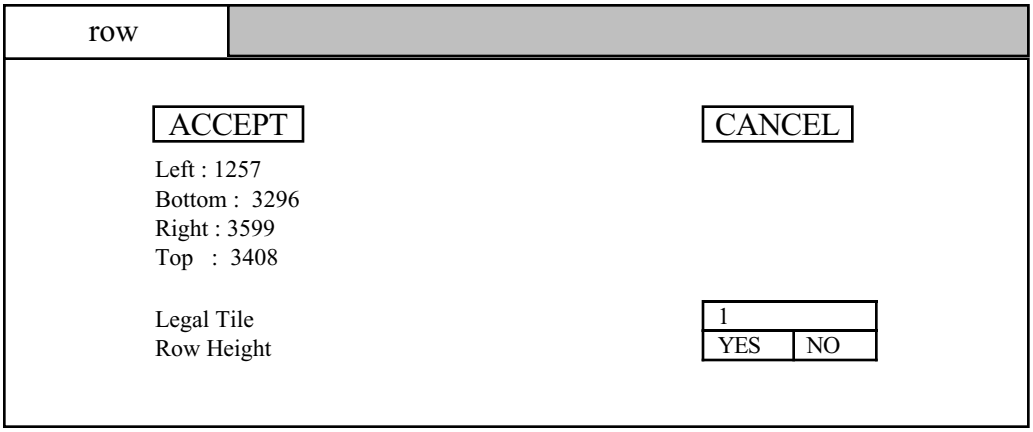


The user may change the coordinates of the macro by either changing the coordinates directly or by adding a delta to the current position. In the latter case, Genrows will update the coordinates automatically. To modify the orientation of the macro, click the mouse pointer in the window which contains the desired orientation. Genrows will respond by rotating the macro in the draw window to the chosen orientation. If the user wishes to cancel the selection, click on REJECT; otherwise, click on the ACCEPT button.

---

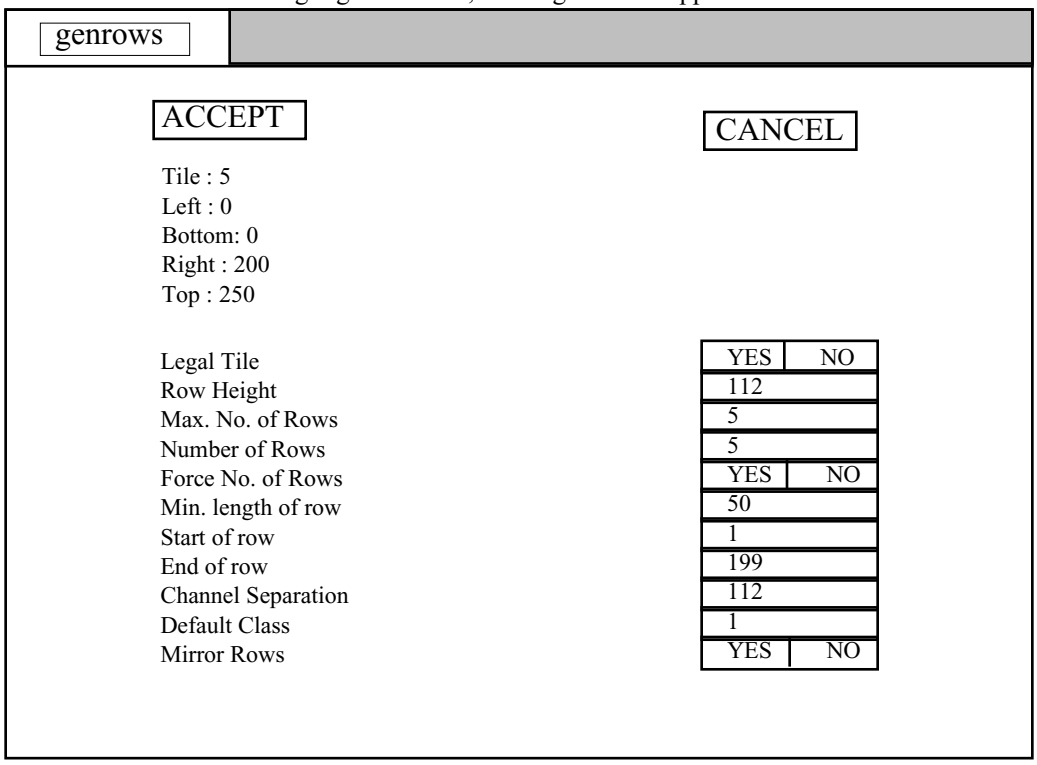
[1]Deactived during Tomus *n*-way partitioning.

Edit Row - allows the modification of a row's class and mirror attributes. Select a row with the mouse pointer. The selected row will become highlighted. Next, a dialog box will appear on the screen:



The row attributes should be added only after the row topology has been determined. Note: It is recommended to set the mirror and class attributes for the entire tile rather than to edit individual rows. The mirror and class attributes are described in 5.2.1.

Edit Tile - allows the modification of a tile's attributes. Select a tile with the mouse pointer. The selected tile will become highlighted. Next, a dialog box will appear on the screen:



At the top of the dialog box, the tile name and dimensions are given. The first tile parameter is the *Legal Tile* switch. If *Legal Tile* is *YES*, then Genrows will attempt to put rows in this tile; otherwise; no rows will be placed in this tile. The second tile parameter is the height of the row. Each tile may have a different row height, but the default is determined from the average

standard cell height. If you need only a small number of rows to have a different row height, you may need to divide a tile into parts. See DIVIDE TILE LEFT_RIGHT below. The parameter *Max. No. of Rows* sets the channel separation in this tile based on the height of the tile and value of this parameter. The tile will now be constrained to place no more than *Max. No. of Rows* within its region; however, it may place less rows if insufficient row length is available. The field *Number of Rows* contains the actual number of rows placed in this tile. Since Genrows places rows from the bottom of the core region upwards, only the top tile containing rows may encounter the situation where *Max. No. of Rows* does not equal *Number of Rows*. If the *Force No. of Rows* switch is set to YES, Genrows will use the *No. of Rows* value to set the number of rows in this tile, maintaining the row separation determined by *Max. No. of Rows*. The user may force the inequality condition *Max. No. of Rows* does not equal *Number of Rows* in any tile by using the *Force No. of Rows* switch. The *Min. length of Row* parameter is used as a criteria for determining whether to place rows in this tile; if the tile width is less than *Min. length of Row*, no rows will be occur in this tile. The *Start of Row* and *End of Row* parameters adjust the beginning and end of all the rows in the tile. All rows must begin and end within the tile. The distance between adjacent rows may be specified using *Channel Separation*. The default class and mirror attributes for all rows in this tile may be set using *Default Class* and *Mirror Rows* respectively.

{**Keep Short Row** | **Discard Short Row**} - toggles between keeping the last *short* row (a row less than the tile width) if it exists and removing any *short* rows. Genrows will round to the row which makes the placed row length closest to the total row length desired.

Modify Core Area[2] - resizes the core area. The user must pick the lower left corner of the core region followed by the upper right corner of the core region. The core region must be at least as large as the minimum bounding box containing all macro cells.

Numrows[3] - modifies the number of standard cell rows. Genrows will divide the total row length evenly between all of the rows.

Redo - returns to the state which was undone in the last step.

Restore State - returns to a previously saved row configuration. Any number of saved configurations are possible but all configuration names must be unique. Genrows will prompt for state name; enter it in the message window. States may be retrieved from previous TimberWolf executions if the total row length and macros in the design remains constant between runs.

Save State - saves the current row configuration. This configuration may later be retrieved using RESTORE STATE. Genrows will prompt for a state name; enter it in the message window. If state name is not unique, Genrows will replace the contents of the named state save file.

Undo - cancels the effect of the last operation. The UNDO command itself can be canceled by the REDO command.

---

[2]Deactived during Tomus *n*-way partitioning.
[3]Available only in standard cell only designs.

Merge Menu

Divide Tile Left_Right - splits a tile into two pieces along a vertical cut line. The dividing line is determined by the mouse pointer.

Divide Tile Up_Down - splits a tile into two pieces along a horizonal cut line. The dividing line is determined by the mouse pointer.

{**Limit Merges** | **Unlimit Merges**} - toggles between limiting tile merge operations to adjacent tiles and allowing the maximum number of tiles to be merged in a single merge operation.

Merge Downward - attempts to merge tiles below the selected tile. The width of tiles below the selected tile must be equal to or greater than the selected tile in order for the merge to proceed. The *merged* tile expands downward, but the width of the tile remains constant.

Merge Left - attempts to merge tiles to the left of the selected tile. The height of tiles to the left of the selected tile must be equal to or greater than the selected tile in order for the merge to proceed. The *merged* tile expands to the left, but the height of the tile remains constant.

Merge Right - attempts to merge tiles to the right of the selected tile. The height of tiles to the right of the selected tile must be equal to or greater than the selected tile in order for the merge to proceed. The *merged* tile expands to the right, but the height of the tile remains constant.

Merge Upward - attempts to merge tiles above the selected tile. The width of tiles below the selected tile must be equal to or greater than the selected tile in order for the merge to proceed. The *merged* tile expands upward, but the width of the tile remains constant.

Reset Tiles - returns the tile configuration to the default maximally horizontal configuration, that is, the tiles will be constructed so that their width is maximized.

Draw Menu

{**Draw Labels** | **Ignore Labels**} - toggle for drawing the labels for macros, tiles, rows, etc.

{**Draw Macros** | **Ignore Macros**} - toggle for drawing the macro cells.

{**Draw Rows** | **Ignore Rows**} - toggle for drawing the standard cell rows.

{**Draw Tiles** | **Ignore Tiles**} - toggle for drawing the core region tiles.

Draw Parameters (these parameters may be set in the *circuitName.par* file)

Feed Percentage - specifies the amount of space to be reserved for feedthrough cells. The amount of cell width reserved will be *feed_percentage* multiplied by the total width of the row-based cells. TimberWolfSC reports the feed percentage of the current execution at the bottom of the circuitName.out file if global routing has been requested. Feed percentage is a floating point number in the range [0.0-100.0].

Min. Row Length - sets a limit on the size of a valid tile, that is, any tile whose width is smaller than the *Min. Row Length* will not have rows. All tiles are set to this value.

Row Separation - represents the desired amount of separation between rows. The amount of separation between rows is this number times the average height of the rows. This is, if you want the row separation equal to the average row height, then this number should be 1.0. On the other hand, if you want the row separation to be twice the height of the rows, then this number should be 2.0. Normally, a value of 1.0 is appropriate.

Set Spacing - constrains the distance between the edge of a tile and the beginning and end of a row.

13. MICKEY

13.1. FUNCTION

13.2. INPUT

13.3. OUTPUT

13.2. GRAPHICAL INTERFACE

14. MINCUT - STANDARD CELL CLUSTERING

14.1. FUNCTION

The Mincut program partitions the standard cells in the netlist into standard cell clusters for floorplanning. These clusters have variable aspect ratios, and in addition, the signal pin locations are not fixed but may be adjusted in order to minimize the wirelength. The Mincut program features a new objective function which better enables the program to find natural partitions of the standard cells [CS88]. The Mincut program partitions the standard cells into standard cell clusters which are approximately equal to the average macro cell area. This insures that both macro cells and standard cell clusters have equal importance in the determination of the core region topology. The area of a standard cell cluster is given by:

$$A_s = (r+1)\sum_{i=1}^{n_s} A_c(i)$$

where

A_s = area of standard cell cluster s

r = the ratio of the row separation distance and the row height

n_s = number of standard cells in cluster s

$A_c(i)$ = area of standard cell i

The estimated row separation factor r accounts for the standard cell routing area needed between the rows. This parameter (**rowSep**) is user supplied.

After partitioning, the Mincut program outputs a netlist for the floorplanner containing the macros and the standard cell clusters. An example of a mixed design after partitioning is shown in Figure 14.1.1 Cells C1-C46 are standard cell clusters. The macro cells (C47-C49) are arbitrarily placed at the origin and the pads (C50-C77) are placed outside the estimated core region.

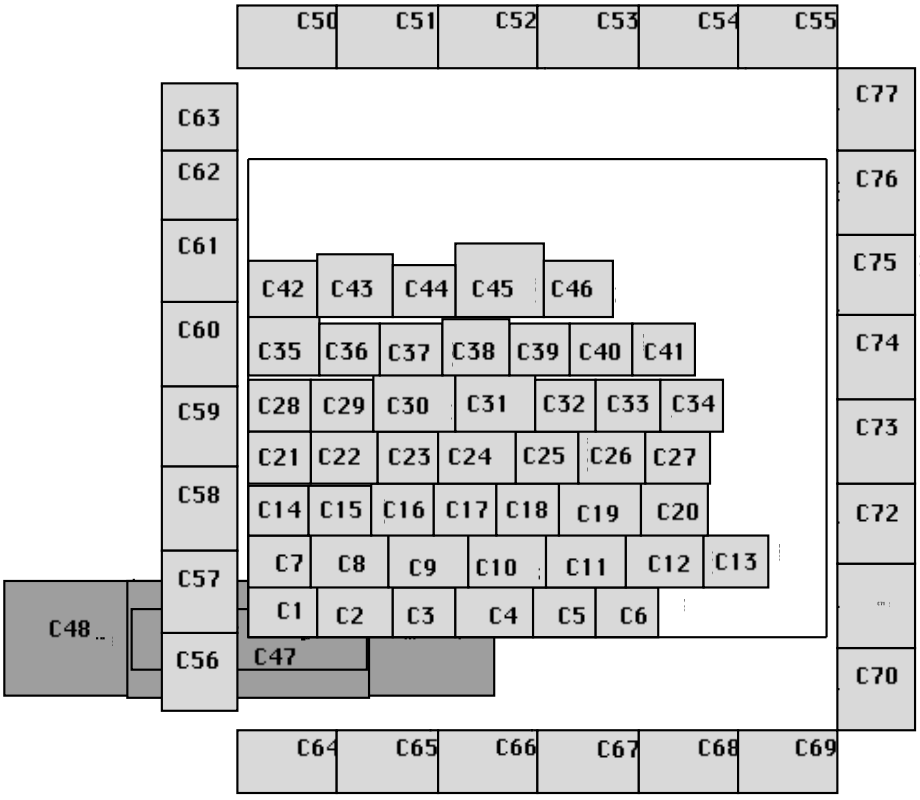


Figure 14.1.1 Result of the *Mincut* partition for a mixed macro/standard cell example.

14.2. INPUT

The input file is *circuitName.cel* which has already been described.

14.3. OUTPUT

The output files are *circuitName.mcel* which is fed to the floorplanner TimberWolfMC and *circuitName.scel* which is input to the row-based placement and global routing tool TimberWolfSC.

14.4. GRAPHICAL INTERFACE

The Mincut program has no graphical interface.

15 PSC

15.1. FUNCTION

PSC schedules TimberWolfSC to run on *n* workstations simultaneously. The program follows Tomus in the partitioning flow. PSC has the knowledge of the number of partitions and other circuit parameters from the flow and attempts to schedule and run TimberWolfSC to place the standard cells of the *partitions*. If the graphics mode is on, the user can watch the graphics window running TimberWolfSC on *n* sub-chips of the original circuit on the *n* workstations he has specified as available in the input file *nodes*. This program is in the developmental stage and hence the input and output files are not final. The following sections will describe its current features.

15.2. INPUT

PSC will schedule TimberWolfSC on the user's current workstation for all *n* sub-chips by default. The user can list at most *n nodes* or workstations available for PSC, a file called nodes. All the other input files required to run TimberWolfSC will be created by Tomus automatically.

15.3. OUTPUT

PSC will create output files for each sub-chip for error-log, output messages and the regular TimberWolfSC output files. The error-log file for each *partition i* on each *node*, is *circuitName*:*i.node*.errorlog. The output file of PSC for each *partition i* on each *node*, is *circuitName*:*i.node*.out.

15.4. GRAPHICAL INTERFACE

At the very beginning of execution, PSC will enter the graphics wait loop. When in the loop, PSC will wait for the user to click on one of the top menu fields in the menu window. The pulldown menu for that entry in the menu window will then become visible. The user then clicks in the desired box for the desired option. PSC will execute the option and return to the graphics wait loop. To continue the program and leave the wait loop, the user must click on the CONTROL menu heading, and then click on the CONTINUE PGM entry. The user may cancel any menu by hitting the CANCEL box in that menu. Figure 15.4.1 shows the contents of all the pulldown menus.

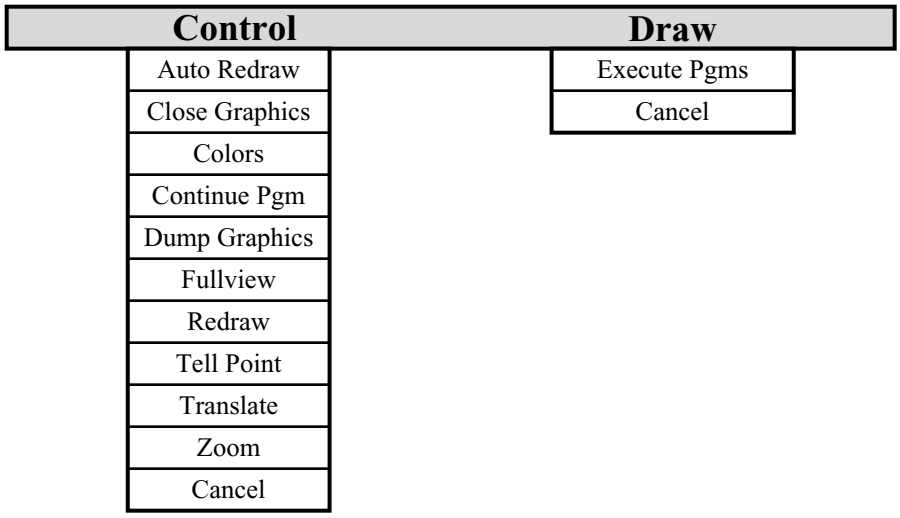


Figure 15.4.1 PSC menu options

The **Control Menu** has the same functions as TimberWolf's other programs. Please refer to the earlier sections for reference.

Edit Menu

Execute Pgms allows the user to start executing the scheduled processes on the *nodes* specified as shown in the graphics window. A sample of the graphics window in shown is Figure 15.4.2.

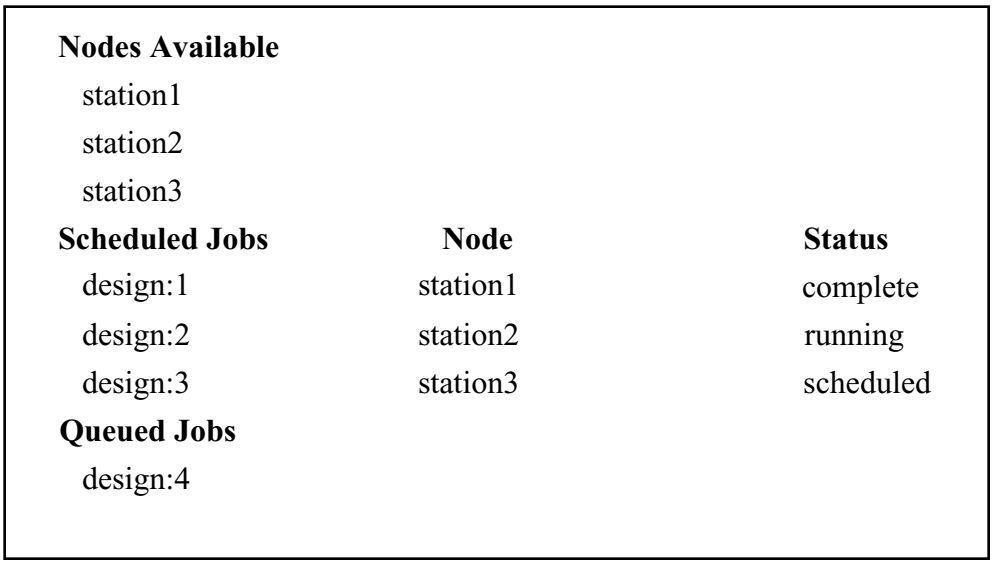


Figure 15.4.2 Sample PSC graphics window

16. SGGR.

17. SYNTAX.

17.1. FUNCTION

This program is incorporated into the twflow program. Its function is to check for errors in the user's input. It is implemented using a yacc/lex parser. This program also compiles vital statistics on the design such as number of standard cells, number of macro cells, etc. to make it easier for downstream programs to allocate resources.

17.2. INPUT

The input file is *circuitName.cel* which has already been described.

17.3. OUTPUT

There are no user output files. An internal system file is generated, the *circuitName.stat* file, but the format of this file is subject to change.

17.4. GRAPHICAL INTERFACE

The syntax program has no graphical interface.

18. TIMBERWOLF (TWFLOW) - THE MASTER CONTROL

18.1. FUNCTION

The sequence of place and route programs is controlled by the master flow program called *twflow*. The program *twflow* creates a flow chart in the graphics window (as shown in Figure 18.1.1) and automatically shows the current status of the layout process by highlighting the current node in the flow chart. A graph is used to describe the design flow. The programs or shell scripts in the flow are nodes in the graph (flowchart) and the edges between the nodes are valid program execution sequences. The program uses file dependencies in order to determine if a program or shell script needs to be executed, similar in function to the Unix *make*.

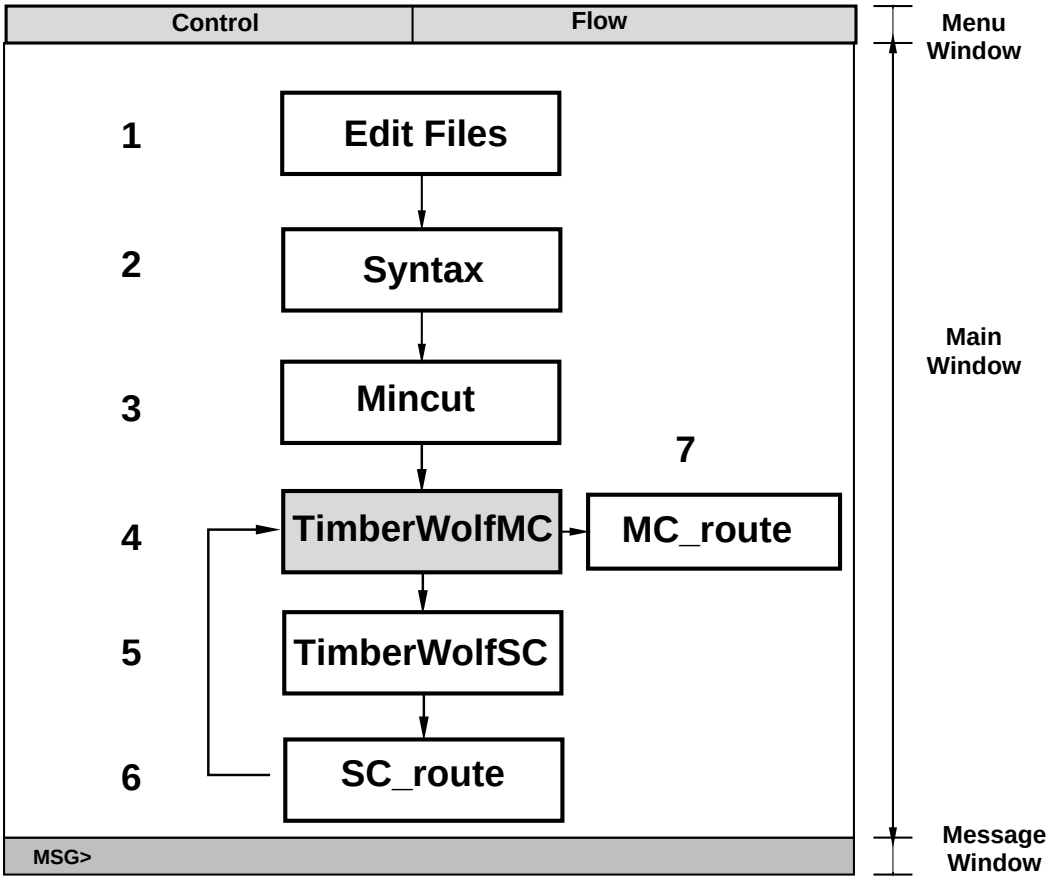


Figure 18.1.1. The TimberWolf interface.

The program *twflow* controls not only the sequence of programs to be executed but also functions as the master control for the X11 graphics. Figure 18.1.1 also shows the X-window interface maintained by the *twflow* program. It consists of three windows: the menu window which controls the pull-down menus, the main window where the current state of the design and/or flow is depicted, and the message window which queries and informs the user of the current status of the design.

The intent was to design a simple interface that was flexible, maintainable, and portable. Flexibility was achieved by not allowing any program to use X11 routines directly. Instead, all graphics requests are handled by an intermediate layer of routines which handle the translation between user coordinates and pixel coordinates. These routines perform the actual requests to the X server. In this way, all programs are

written in the user's coordinate system regardless of graphics system used. If another graphics package becomes available, only the library of intermediate functions needs to be changed.

18.2. INPUT

In order to make the system flexible from the user's perspective, *twflow's* sequence of programs is not predefined; instead, it is defined by an ASCII input file. A portion of the *twflow* input format is shown below.

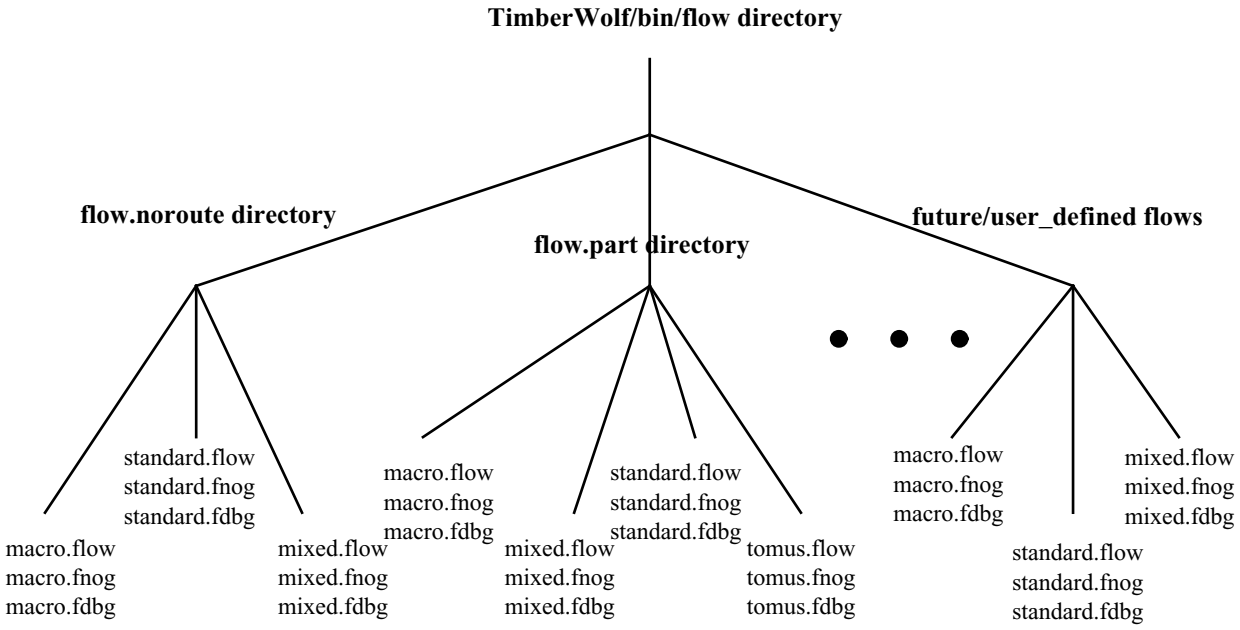
```
numobjects 7
        .
        .
        .
pobject TimberWolfMC 4: 3 6
path :
drawn : 450 450 750 550
edge 3:
ifiles: $.mcel $.mpar $.mnet*
ofiles: $.mdat $.blk
args : -ws $ @WINDOWID
drawn :     600 550 600 650          600 550 620 570     600 550 580 570

edge 6:
ifiles: $_io.mcel $_io.mpar
ofiles: $_io.mdat $_io.blk
args : -ws $_io @WINDOWID
drawn :     250 500 450 500          250 100 450 100     250 100 250 500
            430 520 450 500          430 480 450 500
        .
        .
        .
```

Figure 18.2.2. A portion of the *twflow* file

Each program object or **pobject** is numbered as shown in Figure 18.2.2. The program *TimberWolfMC*, for example, is program number 4 and it may be executed in a valid sequence after program 3 or program 6. The path of the program may be changed from its default place and the user may choose how he wishes to draw the object. Next, each edge is specified, that is, how it is to be executed and what files it depends on. The **$** is a special string substitution character which stands for the design name. For each edge we list the input and output files that are required and generated by the program. By looking at the time stamp of each of these files we can determine if this program needs to be executed. In addition, each edge allows different argument lists to be passed to the executed program. The key word **@WINDOWID** tells *twflow* that the child process is capable of inheriting the graphics interface. Optional files are followed by an asterisk. For example, $.mnet* is an optional input file to TimberWolfMC on edge 3.

It is very easy to modify the sequence of programs in order to customize it to particular applications; for example, another flow file could be generated which allows the addition of front-end and back-end translation programs or perhaps another would omit the **@WINDOWID** key words for batch mode processing.

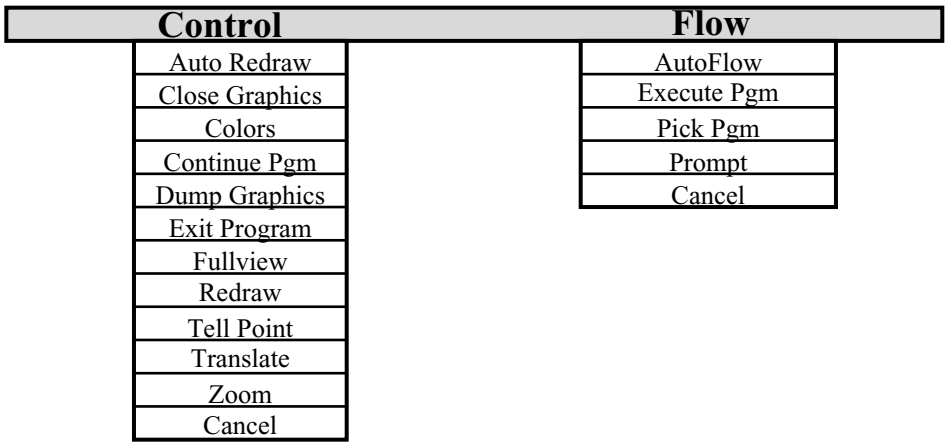
Twflow recognizes two distinct flow file hierarchies: a directory exists for each high level design strategy, and a file exists for each design style (standard, mixed, and macro) within that strategy. In addition, at the design style level, one flow file per style is provided for graphics mode (.flow), no graphics mode (.fnog), and debug mode (.fdbg). Two examples of high level strategies are the basic placement and global routing strategy, *flow.noroute*, and the *n*-way partitioning strategy, *flow.part*. Figure 18.2.3 graphically depicts the flow file structure.



18.3. OUTPUT

The output of Twflow is the *circuitName.stat* file which contains design statistics used by other programs for efficient allocations of resources.

18.4. GRAPHICAL INTERFACE



Control Menu

{**Auto Redraw On** | **Auto Redraw Off**} - toggles between automatic redraw when part of the screen has been covered (on) and suppresssion of this feature (off).

Close Graphics - at any time the user can close the windows and return to batch mode.

Colors - individual colors may turned off or on using a dialogbox.

Continue Pgm - breaks the graphics wait loop and continues execution of the program.

Dump Graphics - dumps the contents of the screen to files in the directory defined by the environment variable DATADIR defined in the current cshell. The Draw program can be used to display this data at a later date.

Exit Program - exit the Twflow program.

Fullview - returns the user to full view of the main window.

Redraw - refresh the display.

Tell Point - returns the user coordinate under the mouse pointer when any mouse button is clicked in the main window.

Translate - performs a reorigin of the main window. The coordinate under the mouse pointer when any mouse button is clicked in the main window will become the new center of the main window.

Zoom - allows the user to zoom in or out to any part of the screen. The zoom (in) area is accomplished by either picking the lower left and upper right points of the zoom rectangle or by entering them in the message window. The points are entered as two integers separated by commas, i.e. 20, 40 for the point (20,40). Note: the pointer must be in the message window to enter points through the keyboard. If the points are entered in the reverse order (upper right, lower left), a zoom out will be performed.

Flow Menu

Autoflow - determines which programs need to be executed based on the input and output file time stamps, and calls each program in turn, passing control of the graphics window to the called program.

Execute Pgm - executes a single program and returns. Program needs to be selected using PICK PGM.

Pick Pgm - select the program to be executed with EXECUTE PGM.

{**Prompt On** | **Prompt Off**} - toggles between prompting the user to decide which path to take when an *out of date* program has multiple execution paths and always taking the first *out of date* path given in the flow file.

19. TIMBERWOLFMC

19.1. FUNCTION

TimberWolfMC is a timing driven floorplanner which handles cells of any rectilinear shape [Sec88]. Furthermore, the cells may have fixed geometry including pin locations (hard macros) or the cells may have an estimated area with a specified aspect ratio range and with pins that need to be placed (soft macros). In addition, the cells may have several possible instances, whereby TimberWolfMC is to select the one which is most suitable. Cells may also be grouped hierarchically. Cells and/or cell groups may be restricted to subregions within the core region. Through the use of the X11 graphics interface, the user may interrupt the automatic execution at any time to add region restrictions or to place macros at specific locations. TimberWolfMC also features a sophisticated I/O placement algorithm which places the pad cells so as to minimize wirelength subject to side, spacing and grouping constraints.

Both TimberWolfMC and the standard cell placement program TimberWolfSC support timing driven placement. The length of the signal paths are forced to lie within user specified bounds. Below is an example of a critical path where A, B, and C are the names of the signals comprising the path, and 0 and 2500 are the lower and upper bounds for the length of the path, respectively.

path A B C : 0 2500

TimberWolfMC insures that the sum of the lengths of the individual signals (*A*, *B*, and *C*) meet the given bounds avoiding the need for the user to partition the path length between the individual signals of the path. Other timing driven placements schemes have been proposed [ML89][OI86][HNY86] but none of them are simulated annealing based and each of these schemes dealt with timing constraints on individual nets rather than signal paths. The use of timing driven placement with simulated annealing was first suggested for gate arrays by de Forcrand et al.[dFZ87] but it was not mentioned in the paper whether the time constraints were incorporated into the cost function. The critical path method is superior to net weighting techniques because it overcomes the partitioning problem and reflects more accurately the true timing constraints to be satisfied.

TimberWolfMC uses a dynamic interconnect-area estimator in order to allocate the necessary interconnect space around each cell [Sec88]. Failure to account for the correct amount of routing area during floorplanning results in placements that require significant placement modification during detailed routing. In this algorithm, we are using the floorplanner to determine the relative placement of the macro cells to the standard cell clusters. If large amounts of placement modification are required during detailed routing, this relative placement will be destroyed. TimberWolfMC updates its estimate of the core region periodically during the course of an execution of the simulated annealing algorithm. The core area estimate in this case is the sum of the macro cell area, the estimated macro cell routing area, and the area of the standard cell clusters. Standard cell clusters already have routing area added to them.

At the completion of the simulated annealing run, the floorplanner has determined the placement of the macro cells. The result of floorplanning is shown in Figure 19.1.1.

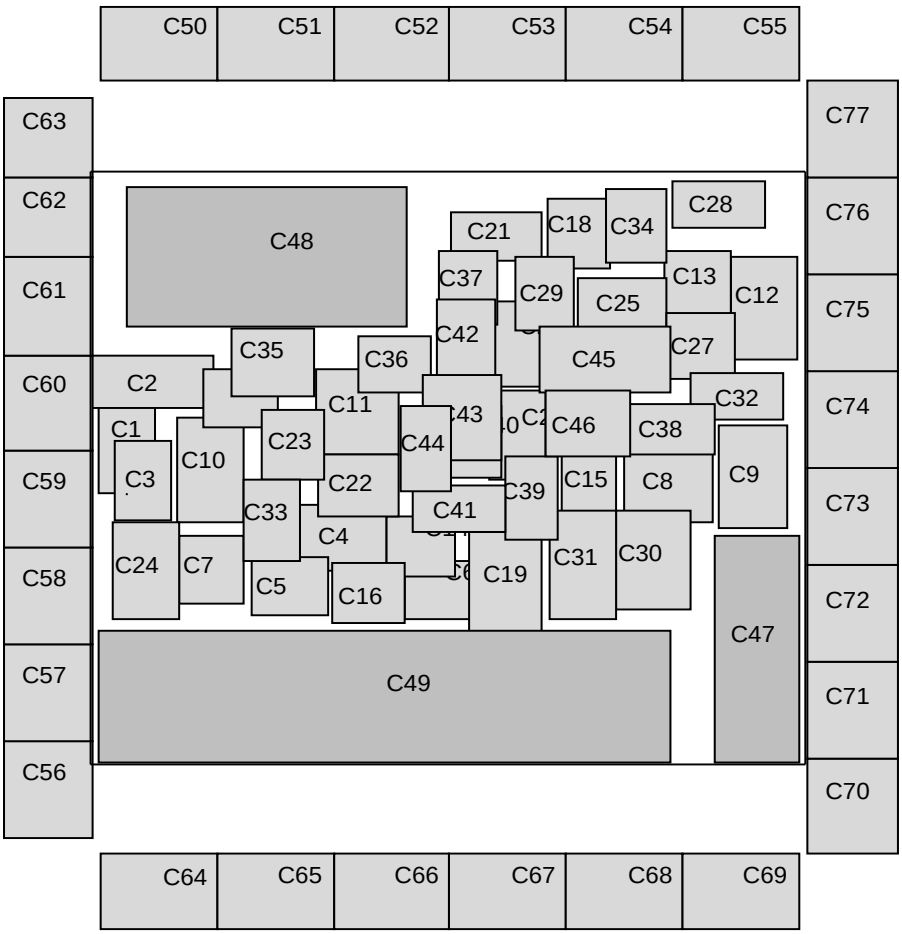


Figure 19.1.1. Result of floorplanning.

19.2. INPUT

The inputs to TimberWolfMC are the *circuitName.cel, circuitName.par, and circuitName.net* files previously described in the TimberWolf input file section.

19.3. OUTPUT

The output from TimberWolfMC is the *circuitName.mdat* file described in the TimberWolf output file section.

19.4. GRAPHICAL INTERFACE

TimberWolfMC allows the user to change various parameters during a graphics wait loop. If **no.graphics.wait** has not been specified, TimberWolfMC will enter graphics wait loops at various logical times during the program - after initial placement, after final placement, and after compaction. In addition, the user may interrupt the progression of the simulated annealing algorithm at any time after initialization and enter into a wait state by clicking the mouse pointer in the menu window. When in the loop, TimberWolfMC will wait for the user to click on one of the top menu fields in the menu window. The pulldown menu for that entry in the menu window will then become visible. The user then clicks in

the desired box for the desired option. TimberWolfMC will execute the option and return to the graphics wait loop. To continue the program and leave the wait loop, the user must click on the **Control** menu heading, and then click on the **Continue Pgm** entry. The user may cancel any menu by hitting the **Cancel** box in that menu.

Figure 19.4.1 shows the contents of all the pulldown menus. Below we explain the meaning of the menu items.

Menu Window

Control	Edit	Draw	Parameters
Auto Redraw	Cell Neighborhood	Draw Bins	Change Aspect Ratio
Close Graphics	Edit Cell	Draw Border	Graphics Wait
Colors	Fix Cell	Draw Globe Areas	Cancel
Continue Pgm	Fix Cell but Rot.	Draw Labels	
Dump Graphics	Group Cells	Draw Neighborhood	
Fullview	Move Cell	Draw Nets	
Graphics Update	Cancel	Draw Pins	
Redraw		Draw Single Net	
Tell Point		Draw Wiring Est	
Translate		Cancel	
Zoom			
Cancel			

Figure 19.4.1. TimberWolfMC graphical menus.

Control Menu

{**Auto Redraw On** | **Auto Redraw Off**} - toggles between automatic redraw when part of the screen has been covered (on) and suppresssion of this feature (off).

Close Graphics - at any time the user can close the windows and return to batch mode.

Colors - individual colors may turned off or on using a dialogbox.

Continue Prog. - breaks the graphics wait loop and continues execution of the program.

Dump Graphics - dumps the contents of the screen to files in the directory defined by the environment variable DATADIR defined in the current cshell. The Draw program can be used to display this data at a later date.

Fullview - returns the user to full view of the main window.

{**Graphics Update On** | **Graphics Update Off**} - toggles between automatic redraw after each adjustment of the cost function parameters (on) and the inhibition of the redraws (off). Interrupt capability remains available in this mode.

Redraw - refresh the display.

Tell Point - returns the user coordinate under the mouse pointer when any mouse button is clicked in the main window.

Translate - performs a reorigin of the main window. The coordinate under the mouse pointer when any mouse button is clicked in the main window will become the new center of the main window.

Zoom - allows the user to zoom in or out to any part of the screen. The zoom (in) area is accomplished by either picking the lower left and upper right points of the zoom rectangle or by entering them in the message window. The points are entered as two integers separated by

commas, i.e. 20, 40 for the point (20,40). Note: the pointer must be in message window to enter points through the keyboard. If the points are entered in the reverse order (upper right, lower left), a zoom out will be performed.

Edit Menu

Cell Neighborhood - set the neighborhood for the selected cell. The cell's center will be constrained to remain within the specified box. Use the mouse to pick the two corners of the box.

Edit Cell - allows the user to edit a macro cell's attributes. Use the mouse pointer to select the desired macro. A dialog box will appear on the screen. Currently, the user may set the xcenter, ycenter, orientation, and valid orientations of the selected macro. If satisfied, hit the ACCEPT button at the top of the dialog box; otherwise, hit the CANCEL button.

Fix Cell - fix the selected cell at its current location and fix its orientation to be the current orientation.

Fix Cell but Rot. - fix the selected cell at its current location but allow orientation changes.

Group Cells - not implemented currently.

Move Cell - moves a selected cell to a new location. Use the mouse pointer to establish a reference point on the selected cell. The cell will then follow the mouse pointer until the mouse button is depressed at the desired location.

Draw Menu

{**Draw Bins** | **Ignore Bins**} - toggle for drawing the bins used during overlap calculation.

{**Draw Border** | **Draw Tiles**} - toggles between drawing the arbitrary rectilinearly shaped cell (border) or the tile decomposition of the rectilinearly shaped cell.

{**Draw Globe Areas** | **Ignore Globe Areas**} - toggles between drawing the spaced needed for routing as determined by the global router or not.

{**Draw Labels** | **Ignore Labels**} - toggle for drawing the labels for cells, edges, nets, etc.

{**Draw Neighborhood** | **Ignore Neighborhood**} - toggle for drawing the cell neighborhoods.

{**Draw Nets** | **Ignore Nets**} - toggle for drawing the nets of the design.

{**Draw Pins** | **Ignore Pins**} - toggle for drawing the pins of the design.

Draw Single Net - draws the minimum spanning tree for a single net. The net is specified by its net number which is output in the circuitName.mpth file. To eliminate this net, either use **Draw Single Net** to change to a new net or toggle **Draw Nets** to **Ignore Nets**.

{**Draw Wiring Est** | **Ignore Wiring Est**} - toggle for drawing the routing area estimated by TimberWolfMC.

Parameters Menu

Change Aspect Ratio - changes the aspect ratio of the design. Type the new aspect ratio into the message window. Make sure the mouse pointer is in the message window.

{**Graphics Wait** | **No Graphics Wait**} - toggle between stop for graphics loops and ignore graphics loops.

20. TIMBERWOLFSC - STANDARD CELL PLACEMENT AND GLOBAL ROUTING

20.1. FUNCTION

After the configuration of the standard cell rows, the placement of the standard cells takes place with the invocation of the *TimberWolfSC* program [SBS85] [SL87]. *TimberWolfSC* uses simulated annealing to place the standard cells by minimizing the total interconnect length subject to timing constraints. Since *TimberWolfMC* has determined the placement of the macro cells relative to the standard cell clusters, we now take the positions of the macro cell pins as fixed and allow the standard cells to be moved so to minimize the total interconnect length. *TimberWolfSC* also performs global routing, that is, it assigns each net segment (pin-to-pin interconnection) to a particular channel on the chip subject to interconnect length and congestion constraints.

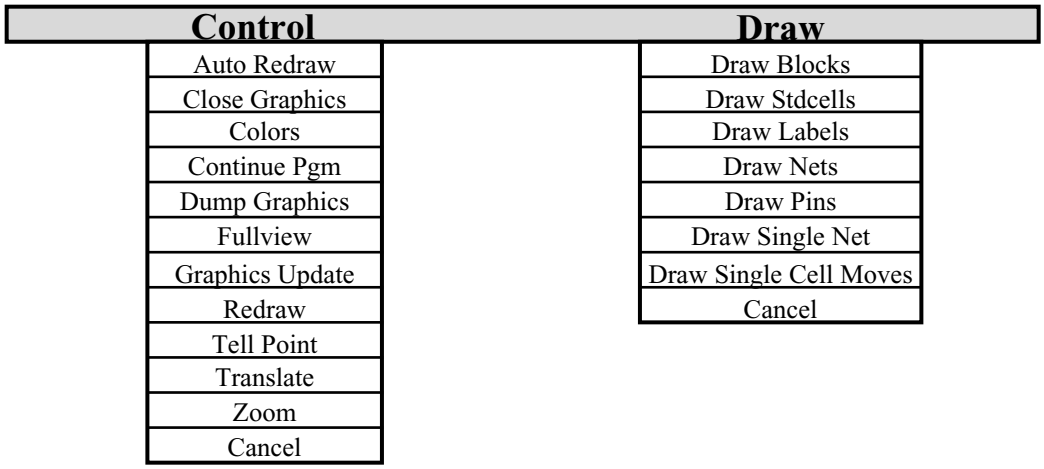
20.2. INPUT

The inputs to TimberWolfSC are the *circuitName.cel, circuitName.par, and circuitName.net* files previously described in the TimberWolf input file section.

20.3. OUTPUT

The output from TimberWolfSC are the *circuitName.pl1, circuitName.pl2, and circuitName.pin* file described in the TimberWolf output file section.

20.4. GRAPHICAL INTERFACE



Control Menu

{**Auto Redraw On** | **Auto Redraw Off**} - toggles between automatic redraw when part of the screen has been covered (on) and suppresssion of this feature (off).

Close Graphics - at any time the user can close the windows and return to batch mode.

Colors - individual colors may turned off or on using a dialogbox.

Continue Pgm - breaks the graphics wait loop and continues execution of the program.

Dump Graphics - dumps the contents of the screen to files in the directory defined by the environment variable DATADIR defined in the current cshell. The Draw program can be used to display this data at a later date.

Fullview - returns the user to full view of the main window.

{**Graphics Update On** | **Graphics Update Off**} - toggles between automatic redraw after each adjustment of the cost function parameters (on) and the inhibition of the redraws (off). Interrupt capability remains available in this mode.

Redraw - refresh the display.

Tell Point - returns the user coordinate under the mouse pointer when any mouse button is clicked in the main window.

Translate - performs a reorigin of the main window. The coordinate under the mouse pointer when any mouse button is clicked in the main window will become the new center of the main window.

Zoom - allows the user to zoom in or out to any part of the screen. The zoom (in) area is accomplished by either picking the lower left and upper right points of the zoom rectangle or by entering them in the message window. The points are entered as two integers separated by commas, i.e. 20, 40 for the point (20,40). Note: the pointer must be in message window to enter points through the keyboard. If the points are entered in the reverse order (upper right, lower left), a zoom out will be performed.

Draw Menu

{**Draw Blocks** | **Ignore Blocks**} - toggle for drawing the desired standard cell rows.

{**Draw StdCells** | **Ignore StdCells**} - toggle for drawing the row-based cells.

{**Draw Labels** | **Ignore Labels**} - toggle for drawing the labels for cells, edges, nets, etc.

{**Draw Nets** | **Ignore Nets**} - toggle for drawing the nets of the design.

{**Draw Pins** | **Ignore Pins**} - toggle for drawing the pins of the design.

Draw Single Net - draws the minimum spanning tree for a single net. The net is specified by its net number which is output in the circuitName.pth file. To eliminate this net, either use **Draw Single Net** to change to a new net or toggle **Draw Nets** to **Ignore Nets**.

{**Draw Single Cell Moves** | **Ignore Single Cell**} - toggles between drawing every accepted move in the annealing process and suppression of this feature.

21. TOMUS

21.1. FUNCTION

Tomus (meaning *cut* in Latin) is our *n*-way circuit partitioning program. It is integrated in the TimberWolf package and is a part of *flow.part*. In *flow.part*, TimberWolfMC finalizes the placement of the macros and the I/O pads and also estimates the core area. Tomus follows from this point. Through the use of the X11 graphics interface, the user can define the physical boundaries of *n* partitions, into which the user desires to divide a large circuit to place and route hierarchically instead of a flat design. Simulated annealing is used to find an optimal set of standard cell clusters in each of these *n* partitions. It minimizes total wire length and optimizes the clusters based on capacity constraints, congestion and timing constraints. In effect, Tomus has a sense of distance minimization and thus minimizes wire length and critical paths in addition to pin-outs on external edges of partitions and satisfying capacity constraints. At the end of the annealing process, Mickey is used to generate optimal routes of the nets of the cells. Tomus generates I/O pads on the edges of all *n* partitions based on these routes. These are called *pseudopads* and corresponding *pseudopadgroups*. Note, at this point the real I/O pads of the main circuit will also be represented in terms of *pseudopads* on the external edges of the main circuit. The user can then interactively call Genrows on each of the partitions to configure the standard cell rows. Each of the *n* partitions will be treated as complete sub-cores at the end of this program. Each of the sub-cores will be placed and routed using our standard cell placement and routing program (TimberWolfSC) , in parallel, using *n* workstations over the network by PSC, the parallel scheduler for TimberWolf.

21.2. INPUT

The input Files Tomus reads are circuitName.mdat and circuitName.mver generated from TimberWolfMC and circuitName.cel, circuitName.ppar and circuitName.nets provided by the user. The circuitName.ppar file is optional. As described in 5.1.4 the parameters used by Tomus can be set in the circuitName.par file which will be then used as default values. But if the user wants to set it in the circuitName.ppar file then the standard format of parameter files should be used as shown below:

fast	*integer*
random.seed	*integer*
rowSep	*float*
slow	*integer*
vertical_path_weight	*float*
vertical_wire_weight	*float*

The description of the above parameters are listed in 5.1.4.

21.3. OUTPUT

The output files Tomus generates are:

for i = 1 to *n*,

circuitName:i.par, circuitName:i.scel and circuitName:i.blk.

The above files are input files required to run TimberWolfSC to place and route standard cells in each of the *n* partitions. Tomus also generates an circuitName.pout file which will record the current status

of Tomus, the runtime errors, the random seed used in the current run, and an update of the values of the cost function at the end of each iteration of simulated annealing during partitioning.

21.4. GRAPHICAL INTERFACE

Tomus will enter graphics wait loops at various logical times during the program - after initial clustering in the partitions, after simulated annealing finds the final clusters, after pseudopads are created, and after Genrows has been called on all *partitions*. When in the loop, Tomus will wait for the user to click on one of the top menu fields in the menu window. The pulldown menu for that entry in the menu window will then become visible. The user then clicks in the desired box for the desired option. Tomus will execute the option and return to the graphics wait loop. To continue the program and leave the wait loop, the user must click on the CONTROL menu heading, and then click on the CONTINUE PGM entry. The user may cancel any menu by hitting the CANCEL box in that menu.

Figure 21.4.1 shows the contents of all the pulldown menus. Below we explain the meaning of the menu items.

Control		Draw
Auto Redraw		Draw Partitions
Colors		Draw Lines
Continue Pgm		Draw Nets
Dump Graphics		Draw Macros
Fullview		Draw StdCells
Redraw		Cancel
Tell Point		
Translate		
Zoom		
Cancel		

Figure 21.4.1. Tomus graphical menus.

Control Menu

{**Auto Redraw On** | **Auto Redraw Off**} - toggles between automatic redraw when part of the screen has been covered (on) and suppression of this feature (off).

Close Graphics - at any time the user can close the windows and return to batch mode.

Colors - individual colors may turned off or on using a dialogbox.

Continue Prog. - breaks the graphics wait loop and continues execution of the program.

Dump Graphics - dumps the contents of the screen to files in the directory defined by the environment variable $DATADIR defined in the current cshell. The DRAW program can be used to display this data at a later date.

Fullview - returns the user to full view of the main window.

Redraw - refresh the display.

Tell Point - returns the user coordinate under the mouse pointer when any mouse button is clicked in the main window.

Translate - performs a reorigin of the main window. The coordinate under the mouse pointer when any mouse button is clicked in the main window will become the new center of the main window.

Zoom - allows user to zoom in or out to any part of the screen. The zoom (in) area is accomplished by either picking the lower left and upper right points of the zoom rectangle or by entering them in the message window. The points are entered as two integers separated by commas, ie. 20, 40 for the point (20,40). Note: pointer must be in message window to enter points through the keyboard. If the points are entered in the reverse order (upper right, lower left), a zoom out will be performed.

Draw Menu

{**Draw Partitions** | **Ignore Partitions**} - toggle for drawing the *partitions* created by the user.

{**Draw Lines** | **Ignore Lines**} - toggle for drawing all the cut lines of partition

{**Draw Nets** | **Ignore Nets**} - toggle for drawing the nets of the design.

{**Draw Macros** | **Ignore Macros**} - toggle for drawing the macros of the design if any.

{**Draw StdCells** | **Ignore StdCells**} - toggle for drawing the standard cells in clusters in the *tiles* of *partitions*.

22. REFERENCES

[CD88]. C. Sechen, and D. Chen, "An Improved Objective Function for Mincut Circuit Partitioning." *Proc. IEEE International Conference on Computer-Aided Design* (1988): 502-505.

[CS90] D. Chen and C. Sechen, "Mickey: a Macro Cell Global Router." submitted to *Int. Workshop on Layout Synthesis*, May 8-11, 1990, MCNC, Research Triangle Park, NC.

[Chi87] M. Chi, "An Automatic Rectilinear Partitioning Procedure for Standard Cell." *Proc. 24th Design Automation Conference* (1987): 50-55.

[dFZ87] Ph. de Forcrand, and H. Zimmermann, "Timing-Driven Auto-Placement," *Proc. Int. Conf on Comp. Design* (1987): 518-521.

[Gro89] P. Groeneveld, "On Global Wire Ordering for Macro-Cell Routing," *Proc. 26th Design Automation Conference* (1989): 155-160.

[HNY86] P. Hauge, R. Nair, and E. Yoffa, "Circuit Placement for Predictable Performance." *Proc. Int. Conf. on Computed-Aided Design* (1987): 88-91.

[KiGV83] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by Simulated Annealing", *Science* 220, 4598, (May 13, 1983), 671-680.

[ML89] M. Marek-Sadowska and S. Lin, "Timing Driven Placement." *Proc. Int. Conf. on Computed-Aided Design* (1989): 94-97.

[OI86] O. Yasushi, T. Ishii, *et al.,* "Efficient Placement Algorithms Optimizing Delay for High-Speed ECL Masterslice LSI's." *Proc. 23rd Design Automation Conference* (1986): 404-410.

[PSS88]. R. Putatunda, D. Smith, M. Stebnisky, C. Puschak, and P. Patent, "VITAL: Fully Automatic Placement Strategies for Very Large Semicustom Designs." *Proc. IEEE International Conference on Computer Design: VLSI in Computers & Processors*: (1988): 434-439.

[SBS85] C. Sechen, D. Braun, and A. Sangiovanni-Vincentelli, "ThunderBird: A Complete Standard Cell Layout Package." *IEEE J. of Solid-State Circuits* 23/2 (1985): 410-420.

[SecS84] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package", *Proc. 1984 Custom Integrated Circuit Conf.*, Rochester, NY, May 1984.

[SecS85] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package", *IEEE J. of Solid State Circuits*, April 1985.

[Sec88] C. Sechen, "Chip-Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing." *Proc. 25rd Design Automation Conference* (1988): 73-80.

[Sec88b] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers (1988).

[SL87] C. Sechen and K. W. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement." *Proc. of ICCAD* (1987): 478-481.

23. APPENDIX A - SYNTAX FOR THE CIRCUITNAME.CEL FILE

Below is the BNF followed by a list of reserved keywords for the *circuitName.cel* file. Note: the BNF may contain more information than described in the manual due to ongoing research. All keywords are capitalized in the BNF.

23.1. BNF for *circuitName.cel* file

```
start_file          : core pads
                    | core

core                : corecells
                    | corecells cellgroups

corecells           : coretype
                    | corecells coretype

coretype            : hardcell
                    | softcell
                    | stdcell

pads                : padcells
                    | padcells padgroups

padcells            : padcell
                    | padcells padcell

padgroups           : padgroup
                    | padgroups padgroup

cellgroups          : cellgroup
                    | cellgroups cellgroup

stdcell             : cellname std_fixed mirror bbox stdgrppins
                    | cellname optional_list std_fixed mirror bbox stdgrppins

optional_list       : option
                    | optional_list option

option              : celloffset
                    | eco
                    | swap_group
                    | legal_block_classes
                    | initial_orient
```

```
hardcell            : hardcellname custom_instance_list
                    | hardcellname fixed custom_instance_list custom_instance_list: custom_instance
                    | custom_instance_list instance custom_instance

custom_instance     : corners class orient hardpins
                    | corners class orient

softcell            : softname soft_instance_list
                    | softname fixed soft_instance_list

soft_instance_list  : soft_instance
                    | soft_instance_list instance soft_instance

soft_instance       : corners aspect class orient softpins mc_pingroup
                    | corners aspect class orient softpins
                    | corners aspect class orient

instance            : INSTANCE string

padcell             : padname corners cur_orient restriction sidespace hardpins
                    | padname corners cur_orient restriction sidespace

padgroup            : padgroupname padgrouplist restriction sidespace

cellgroup           : supergroupname supergrouplist class orient
                    | cellgroupname neighborhood cellgrouplist

hardcellname        : HARDCELL string NAME string

softname            : SOFTCELL string NAME string

cellname            : CELL string string

neighborhood        : NEIGHBORHOOD
                            INTEGER FROM xloc INTEGER FROM yloc
                            INTEGER FROM xloc INTEGER FROM yloc
                    | NEIGHBORHOOD FIXED
                            INTEGER FROM xloc INTEGER FROM yloc
                            INTEGER FROM xloc INTEGER FROM yloc
```

```
fixed                : fixedcontext AT INTEGER FROM xloc INTEGER FROM yloc
                     | fixedcontext NEIGHBORHOOD
                           INTEGER FROM xloc INTEGER FROM yloc
                           INTEGER FROM xloc INTEGER FROM yloc

fixedcontext         : FIXED

std_fixed            : /* empty */
                     | initially fixed_type INTEGER FROM fixed_loc OF
                           BLOCK INTEGER

swap_group           : SWAPGROUP string

celloffset           : CELLOFFSET offset_list

offset_list          : INTEGER
                     | offset_list INTEGER

eco                  : ECO_ADDED_CELL

legal_block_classes  : LEGALBLKCLASS num_block_classes block_classes

num_block_classes    : INTEGER

block_classes        : block_class
                     | block_classes block_class

block_class          : INTEGER

initial_orient       : ORIENT INTEGER

initially            : /* empty */
                     | INITIALLY

fixed_type           : FIXED
                     | NONFIXED
                     | RIGIDFIXED

fixed_loc            : LEFT
                     | RIGHT

mirror               : /* empty */
                     | NOMIRROR
```

```
bbox                : LEFT INTEGER RIGHT INTEGER BOTTOM INTEGER TOP INTEGER

xloc                : STRING

yloc                : STRING

padname             : PAD string NAME string

padgroupname        : PADGROUP string PERMUTE
                    | PADGROUP string NOPERMUTE

supergroupname      : SUPERGROUP string NAME string

cellgroupname       : CELLGROUP string NAME string

corners             : CORNERS INTEGER cornerpts

cornerpts           : INTEGER INTEGER
                    | cornerpts INTEGER INTEGER

class               : CLASS INTEGER

orient              : INTEGER ORIENTATIONS orientlist cur_orient
                    | ORIENTATIONS orientlist cur_orient

orientlist          : INTEGER
                    | orientlist INTEGER

cur_orient          : /* empty */
                    | ORIENT INTEGER

aspect              : ASPLB FLOAT ASPUB FLOAT

softpins            : softtype
                    | softpins softtype

softtype            : pintype
                    | softpin

hardpins            : pintype
                    | hardpins pintype
```

```
stdgrppins          : std_grppintype
                    | stdgrppins std_grppintype

stdpins             : std_pintype
                    | stdpins std_pintype

std_grppintype      : pinrecord
                    | pinrecord equiv_list
                    | pinrecord unequiv
                    | pingroup

std_pintype         : pinrecord
                    | pinrecord equiv_list
                    | pinrecord unequiv

pintype             : pinrecord
                    | pinrecord equiv_list

pingroup            : PINGROUP stdpins ENDPINGROUP

softpin             : softpin_info siderestriction
                    | softpin_info siderestriction softequivs

softpin_info        : SOFTPIN NAME string SIGNAL string opt_layer

pinrecord           : PIN NAME string SIGNAL string layer INTEGER INTEGER

equiv_list          : equiv
                    | equiv_list equiv

equiv               : EQUIV NAME string layer INTEGER INTEGER

unequiv             : UNEQUIV NAME string layer INTEGER INTEGER

softequivs          : mc_equiv
                    | mc_equiv user_equiv_list
                    | user_equiv_list

mc_equiv            : addequiv siderestriction

addequiv            : ADDEQUIV
```

```
user_equiv_list     : user_equiv
                    | user_equiv_list user_equiv

user_equiv          : equiv_name siderestriction connect

equiv_name          : EQUIV NAME string opt_layer

connect             : /* empty */
                    | CONNECT

mc_pingroup         : pingroupname pingrouplist siderestriction
                    | mc_pingroup pingroupname pingrouplist siderestriction

pingroupname        : PINGROUP string PERMUTE
                    | PINGROUP string NOPERMUTE

pingrouplist        : pinset
                    | pingrouplist pinset

pinset              : string FIXED
                    | string NONFIXED

siderestriction     : /* empty */
                    | RESTRICT SIDE side_list

side_list           : INTEGER
                    | side_list INTEGER

sidespace           : /* empty */
                    | SIDESPACE FLOAT

layer               : LAYER INTEGER

opt_layer           : /* empty */
                    | LAYER INTEGER

sideplace           : STRING

restriction         : /* empty */
                    | RESTRICT SIDE sideplace

padgrouplist        : padset
                    | padgrouplist padset
```

```
padset          : string FIXED
                | string NONFIXED

supergrouplist  : string
                | supergrouplist string

cellgrouplist   : string
                | cellgrouplist string

string          : STRING
                | INTEGER
                | FLOAT
```

23.2. Reserved keywords for *circuitName.cel* file

Note: TimberWolf *is* case sensitive.

TOKEN	DEFINITION
ECO_ADDED_CELL	ECO_added_cell
ADDEQUIV	addequiv
ASPLB	asplb
ASPUB	aspub
AT	at
BLOCK	block
BOTTOM	bottom
CELL	cell
CELLOFFSET	cell_offset
CELLGROUP	cellgroup
CLASS	class
CONNECT	connect
CORNERS	corners
ENDPINGROUP	end_pin_group
EQUIV	equiv
FIXED	fixed
FROM	from
HARDCELL	hardcell
INITIALLY	initially
INSTANCE	instance
LAYER	layer
LEFT	left
LEGALBLKCLASS	legal_block_classes
NAME	name
NEIGHBORHOOD	neighborhood
NOMIRROR	nomirror
NONFIXED	nonfixed
NOPERMUTE	nopermute
OF	of
ORIENT	orient
ORIENTATIONS	orientations
PAD	pad
PADGROUP	padgroup
PERMUTE	permute
PIN	pin
PINGROUP	pin_group
RESTRICT	restrict

RIGHT	right
RIGIDFIXED	rigidly_fixed
SIDE	side
SIDESPACE	sidespace
SIGNAL	signal
SOFTCELL	softcell
SOFTPIN	softpin
STDCELL	stdcell
SUPERGROUP	supergroup
SWAPGROUP	swap_group
TOP	top
UNEQUIV	unequiv

Definitions of INTEGER, FLOAT and STRING (as defined by lex).

letter	[!"#$%&'()*+,\-.\/a-zA-Z:;<=>?@\[\\\]^_`{|}~]
digit	[0-9]
alphanum	[!"#$%&'()*+,\-.\/a-zA-Z0-9:;<=>?@\[\\\]^_`{|}~]
new_line	[\n]
blank	[\t]
sign	[-+]
exponent	[eE]
dot	[.]

C comment	"/*""/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"*/"
INTEGER	{sign}?{digit}+
FLOAT	{sign}?{digit}*{dot}{digit}*
FLOAT	{sign}?{digit}+{dot}{digit}*{exponent}{sign}?{digit}+
STRING	{digit}*{letter}{alphanum}* (if not a keyword)