ICARuS

version 1.5

CAD Science, Inc.

www.cadscience.com

1 Introduction

ICARuS (**IC A**dapted **Ru**le **S**ystem) aims for the solution to any rule based layer checking/conversion. User script is Tcl script with **ICARuS** commands procedures. We adapted Tcl instead of creating new script language to add high programmability and adaptability. Program simply sources user script under tclsh. User can write or source Tcl/Tk program inside script for best efficiency as long as there is no name conflict with **ICARuS** commands and subroutines. Note that system checks the syntax of whole script before it actually starts to execute for your convenience because Tcl/Tk is interpreter, not compiler.

Note that user doesn't need to know/learn Tcl language.

We provide node-lock timed licensing on Windows platform while floating timed licensing is provided on Linux/Unix.

1.1 Installation

Note: If you have multiple network cards including wireless, the same network card must be used in installation and running our product (license server only on Linux). If no network card has connection when installing, there should be no network card connection when running.

Windows

1) Install Tcl/Tk if not installed in the system. We recommend to install ActiveTcl from www.tcl.tk.

2) Execute **icarus** package.

3) For manual installation, copy **icarus** directory into certain place. It is installation path. Copy msvcp71.dll, msvcr71.dll, and stlport_vc7146.dll into Windows/system32 directory.

Linux/Unix

1) Untar icarus package.

2) Set environment variable icarus_dir to installed directory (e.g., “setenv icarus_dir install_dir”). Include $icarus_dir in path (e.g., “set path=($icarus_dir $path)”).

1.2 How to Run

Windows - run from DOS window

Basic Syntax: tclsh installPath/icarus.tcl userScript.tcl

- Manual installation

Put icarus path into environment variable (e.g., "set icarus=c:\icarus" from dos).

And then you can run by “tclsh %icarus%\icarus.tcl script.tcl”

- Automatic installation from setup package

If you installed from setup package, simply copy icarus.bat to any path defined (e.g., c:\Tcl\bin). You can check the path by typing “echo %path%” from DOS window.

And then you can run by “icarus script.tcl”.

Linux/Unix - run from console or xterm

You can run by “icarus script.tcl”.

1.3 File Size and 32-bit Limit

On Windows, FAT32 file system can have up to 4GB file while NTFS allows tera byte file.

On Linux, ext2 file system can have up to 2G file. Ext3 file system depends on block size (16G for 1kb, 2TB for 4kb).

Note that normal 32-bit Linux/Unix and Windows versions can read up to 2G file. If GDS file is bigger than 2G, it has to be gzipped to less than 2G and then program can open properly. Converted library (external memory system, any size) can be opened with 32-bit versions as long as memory can hold. The same library format is used on 32-bit and 64-bit versions for compatibility.

Note that our special versions of Windows (large file handling) and Linux can read over 2G file. The only limitation is your file system (FAT32 or NTFS, ext2 or ext3).

1.4 Windows vs. Linux/Unix

- 32-bit Linux/Unix version has memory limitation of 2G to 3.5G depends on kernel while 32-bit Windows version has that of 2G.
- Hierarchical operations will run over 1G GDS data on 32-bit Windows and 1.5G GDS on 32-bit Linux.
- Flat operations will run over 2G GDS on 32-bit Windows and 3G GDS on 32-bit Linux depends on kernel.

2 Library

When program reads GDSII/OASIS file, it converts to our own external memory system (library). Our external memory system files will be placed under user specified directory. All files will be sized up to around 1G and it will be portable to all kinds of platforms (32-bit/64-bit, Windows/Linux/Unix, FAT32/NTFS/ext2/ext3 etc). All platforms share the same library and user can simply copy from one to the other platform.

Library is typically composed of the followings files:

- Info file – “info” binary file, contains some information.
- DB file – “**db***i*” binary file where *i* is number, majority of data will be placed.
- Cell name file – “names” text file, contains all cell names. User can browse this file but the change of file must be cared. User can change/rename cell names in this file but the order of cell names must not be changed (e.g. delete line and reordering are prohibited).
- Parent file – “parents” text file, it shows parent cells for each cell. User can browse relationship but the change of file is prohibited.

3 Commands

3.1 Base Commands

- **LogFile -** set log file

Syntax: LogFile fileName

All screen output messages will be put into user specified log file.

- **Library** - define library directory

Syntax: Library[Z] directoryName

Put Z for gzipped library.

- **SetLibrary** - set current library when using with multiple libraries

Syntax: SetLibrary id

User can load upto 10 libraries in one script. The id is [1..10].

This command can be used inside **ForEachArea**.

- **Areas -** set operation areas

Syntax: Areas { { cellName x1 y1 x2 y2 } ... }

cellName - set "" to process root cell (the first root cell if input is forest)

x1/y1 - left bottom x/y coordinate in nano-meter (*nm*).

x2/y2 - right top x/y coordinate in nano-meter

Set invalid area to process whole cell.

Areas will be reset to new one if previously defined.

If **Areas** is not defined, it will process whole chip.

- **Layers** - set operation layers

Syntax: Layers { { layer datatype } ... }

Note that each layer is defined as "{ layer datatype }", this is the convention in this system.

Do not set this if you want to process whole layers.

Layers will be reset to new one if previously defined.

- **ResetLayers** - void previous layer setting

Syntax: ResetLayers

This will be useful if previous Layers are set but to process whole layers afterwards.

3.2 IO Commands

- **GDSIn** - read GDS file into library

Syntax: GDSIn fileName[.gz] [Prefix]

Put .gz extension for gzipped input. If Prefix is defined, put Prefix to all reading cell names from specified gds file.

If **Layers** are defined, only specified operation layers will be read.

One can read multiple gds into one library by repeating GDSIn. If same cell appears in the later gds file, later cell will overwrite the previous cell in the library. If one reads gds file into existing library, existing library will be reset to NULL and read.

- **OASISIn** - read OASIS file into library

Syntax: OASISIn fileName [Prefix]

If Prefix is defined, put Prefix to all reading cell names from specified gds file.

If **Layers** are defined, only specified operation layers will be read.

One can read multiple oasis files into one library by repeating OASISIn. If same cell appears in the later oasis file, later cell will overwrite the previous cell in the library. If one reads oasis file into existing library, existing library will be reset to NULL and read.

- **LayerMap** - set layer mapping for GDS/OASIS in/out

Syntax: LayerMap { { layer1 datatype1 layer2 datatype2 } ... }

Layers (layer1, datatype1) will be mapped to new one (layer2, datatype2) when it reads/writes to GDS/OASIS.

- **ResetLayerMap** - void previous layer mapping

Syntax: ResetLayerMap

This will be useful if previous LayerMap is set but to process current layers as is.

- **PolyIn** - read polygon text file into the specified layer of given cell

Syntax: PolyIn fileName cellName Layer

Layer syntax is {layer, datatype}.

Polygon file is pure text file having set of polygons and each polygon is represented by "no_of_points x_1 y_1 x_2 y_2 ... x_n y_n" where x_i/y_i is x/y coordinate of i-th point. For rectangle, one may use "0 x1 y1 x2 y2" format for file size reduction where x1/y1 is bottom left point and x2/y2 is top right point.

Note that coordinate is not by *nm*, it is by library's database unit and cell magnifications.

E.g.) temp.poly file has the following lines for two polygons

```
0
100 100
300 300
3
200 200
400 200
300 300
```

3.3 Tech Parameters

Technical detailed parameters can be set before applying commands. Note that all parameter setting commands in this system have “Set” prefix.

- **SetAreaMargin** - set additional margin for processing area and input unit is *nm*

Syntax: SetAreaMargin amount

This parameter is for dimensional layer operation. For example, size down and up needs to consider little bit more on boundary polygons before processing. In **SaveDB**, output will be chopped automatically by original processing area.

If no value is set, it will automatically calculate, report, and use that value. User needs to adjust value if necessary.

- **SetLayer** - to use alphabetic name for layer instead of {layer datatype} pair

Syntax: SetLayer name { layer datatype }

The difference between using normal Tcl set command and SetLayer is that SetLayer is reported in log file reports (e.g., SetLayer poly { 10 0 }).

3.4 Layer Operation Commands

Layer operations are commanded inside **ForEachArea** or **Hierarchical** command set while base commands can be defined outside of those.

3.4.1 Basic Commands

- **ForEachArea** - apply command set for each area (**Areas**)

Syntax: ForEachArea { { command1 ... } { command2 ... } ... } or

ForEachArea { { source commands1.tcl } { source commands2.tcl } … }

It finds the first area and runs each command in the given order.

At the end of commands, it resets all temporary layers and goes to the next area.

It repeats this process for all specified area.

The comment syntax inside this command is {# ... }. If there are "{", there must be matching "}".

Other syntax follows Tcl/Tk syntax for list. For example, if user wants to continue in the next line for one command, user has to put "\" to indicate the next line is still in one list.

If area is too big, program will slice automatically in 2D way by 25x25 *um* (by default) and process each sliced area for **ForEachArea** commands. Note that little bit smaller *y* is better than square shape, this is because of engine behavior. To change default process area unit, set **SetProcessSizeX/Y** (or **SetProcessSize** to set both numbers to one) to desired number in *nm* before **Areas** is defined. The overall performance depends on the number of input and intersecting vertices (i.e., density) which depends on the technology node. User can observe the log to check the number of input vertices on some dense area to be less than a million for fast processing.

E.g.) set **SetProcessSizeX** 100000; set **SetProcessSizeY** 200000

- **TempLayer** - set variable for temporary operation resulting layer

Syntax: TempLayer newName Layer

Layer is { layer datatype }.

Once set, user can reuse specified temporary layer in their complex formular and it won't be erased inside command set.

Note that if you just want to use alphabetic name instead of layer/datatype numbers, you can use **SetLayer** command (e.g., SetLayer poly { 10 0 }).

- **ResetTempLayer -** remove intermediate layer defined by **TempLayer**

Syntax: ResetTempLayer TempLayer

TempLayer is layer name defined by **TempLayer**.

This is for user to control tighter memory management inside **ForEachArea.** User can save memory usage by removing unuseful temp layer at any point instead of waiting for automatic all temporary layer removal at the last stage of **ForEachArea**.

- **ResetTempLayers -** remove all intermediate layers defined by **TempLayer**

Syntax: ResetTempLayers

This command is executed at the end of **ForEachArea** automatically but user can call inside of command sequence instead of individual temp layer reset commands.

- **Append** - append the second temporary layer to the first temporary layer

Syntax: Append Layer1 Layer2

Layer is temporary layer, not the data base layer. Layers contain edges or polygons.

- **SaveDB** - save temporary operation resulting layer into data base layer

Syntax: SaveDB Layer1 Layer2

Layer is { layer datatype }.

Temporary layer will be removed after putting into data base.

If **SetAreaMargin** was called, it will automatically chop temporary layer by original area and save into data base.

- **SavePoly** - save temporary operation resulting layer into polygon text file

Syntax: SavePoly TempLayer FileName

TempLayer is { layer datatype } which is result of **TempLayer**.

FileName is file name to store polygons.

This command is for error checking output instead of layout modification (**SaveDB**).

Temporary layer will be removed after outputting.

If **SetAreaMargin** was used, it will automatically chop temporary layer by original area and save into text file. If multiple areas are processed, all results will be put into single file.

If distributed processing is run, file name will become FileName_id.ext (e.g., space_err_0.poly, …, space_err_3.poly where FileName is space_err.poly and totoal dp is 4). User need to append output text files to create single output file using **MergeTextFiles**. Refer multi CPU section.

Saved file can be read into library by **PolyIn** command.

Note that this command cannot be used with hierarchical operations because it doesn't keep cell information. This is for **ForEachArea** only.

3.4.2 Boolean Commands

Input layers of Boolean commands contain edges or polygons. If EDGE is not specified, output must form a polygon otherwise program will stop.

- **MergeLayers** - apply boolean OR on one or more layers

Syntax: MergeLayers

Input layers are specified by **Layers** command.

Merge multiple layers (removing overlaps), store in temporary layer, and return that temporary layer.

- **OR1** - apply boolean OR on single layer

Syntax: OR1 Layer

Layer is { layer datatype }. Layer contains edges or polygons.

Merge single layer (removing overlaps), store in temporary layer, and return that temporary layer.

If one layer will be used multiple times, it is strongly recommended to assign layer to temporary layer and reuse using **TempLayer**.

Note that all other Boolean operations except **Holes** contain OR1 so it is unnecessary to apply OR1 before applying other Boolean operation.

- **OR** - apply boolean OR

Syntax: OR Layer1 Layer2

Layer is { layer datatype }. Layers contain edges or polygons.

Merge two layers, store in temporary layer, and return that temporary layer.

Note that all boolean operations return temporary layer ({layer datatype}) so that user can write complex formular.

You can also reuse temporary layer by using TempLayer command.

- **AND1** - apply boolean AND on single layer

Syntax: AND1 Layer

Layer is { layer datatype }. Layer contains edges or polygons.

Find overlaps of specified layer, store in temporary layer, and return that temporary layer.

- **AND** - apply boolean AND

Syntax: AND Layer1 Layer2

Layer is { layer datatype }. Layers contain edges or polygons.

Find overlaps of two layers, store in temporary layer, and return that temporary layer.

- **XOR** - apply boolean exclusive-or

Syntax: XOR Layer1 Layer2

Layer is { layer datatype }. Layers contain edges or polygons.

Exclude overlaps from merged shapes of two layers, store in temporary layer, and return that temporary layer.

- **DIFF** - subtract one layer from the other layer

Syntax: DIFF Layer1 Layer2

Layer is { layer datatype }. Layers contain edges or polygons.

Layer1 - Layer2, store in temporary layer, and return that temporary layer.

- **Inverse** - inverse polygons surrounded by working area

Syntax: Inverse Layer

Layer is { layer datatype }.

Holes and background become polygons and original polygons become holes surrounded by area.

Returns resulting layer.

3.4.3 Relational Commands

Input layers of Relational commands contain edges or polygons. If EDGE is not specified, output must form a polygon otherwise program will stop.

- **RELATE** - to create relational DB on two layers

Syntax: RELATE Layer1 Layer2

Layer is { layer datatype }.

Output is temporary layer which will support and speed up relational DB operations having prefix R. Note that relational operation (e.g., boolean operation with prefix R) needs this command to be defined upfront.

Comparing to single boolean operation, this will have better speed in cases where two layer relations are multiply used.

E.g.) applying both AND and DIFF

```
ForEachArea {
      { TempLayer poly_diff {RELATE {1 0} {2 0}} }
      { SaveDB {RAND poly_diff} {101 0} }
      { SaveDB {RDIFF2 poly_diff} {102 0} }
}
```

- **ROR1** - apply boolean OR on the first layer of related layers

Syntax: ROR1 TempLayer

TempLayer is resulting layer of **RELATE** command.

Merge single layer (removing overlaps), store in temporary layer, and return that temporary layer.

If one layer will be used multiple times, it is strongly recommended to assign layer to temporary layer and reuse.

- **ROR2** - apply boolean OR on the second layer of related layers

Syntax: ROR2 TempLayer

TempLayer is resulting layer of **RELATE** command.

- **ROR** - apply boolean OR on related layers

Syntax: ROR TempLayer

TempLayer is resulting layer of **RELATE** command.

Merge two layers, store in temporary layer, and return that temporary layer.

- **RAND1** - apply boolean AND on the first layer

Syntax: RAND1 TempLayer

TempLayer is resulting layer of **RELATE** command.

Find overlaps of specified layer, store in temporary layer, and return that temporary layer.

- **RAND2** - apply boolean AND on the second layer

Syntax: RAND2 TempLayer

TempLayer is resulting layer of **RELATE** command.

- **RAND** - apply boolean AND on related layers

Syntax: RAND TempLayer

TempLayer is resulting layer of **RELATE** command.

Find overlaps of two layers, store in temporary layer, and return that temporary layer.

- **RDIFF** - subtract the second layer from the first layer on related layers

Syntax: RDIFF[2] TempLayer

TempLayer is resulting layer of **RELATE** command.

Layer1 - Layer2, store in temporary layer, and return that temporary layer.

RDIFF2 is to subtract the first layer from the second layer on related layers, i.e., Layer2 - Layer1.

- **RXOR** - apply boolean exclusive-or on related layers

Syntax: RXOR TempLayer

TempLayer is resulting layer of **RELATE** command.

Exclude overlaps from merged shapes of two layers, store in temporary layer, and return that temporary layer.

- **IN** - extract polygons of one layer inside the other layer

Syntax: [S/T]IN[2] TempLayer

TempLayer is resulting layer of **RELATE** command.

There are three kinds of inside: SIN (strict inside), TIN (touch inside), IN (inside).

SIN is to extract polygons of the first layer defined by **RELATE** which are strict inside (no touching) of the second layer defined by **RELATE**.

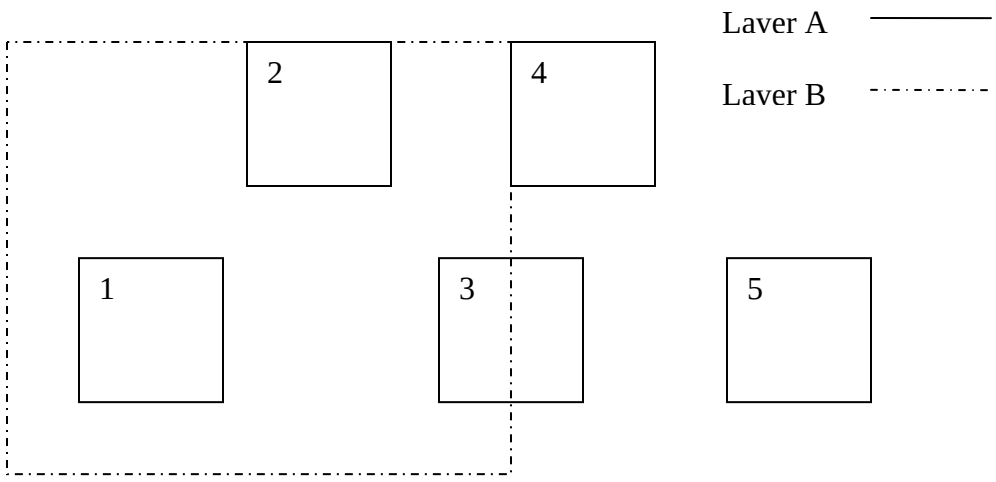
TIN is to extract polygons of the first layer which are inside of the second layer but have edges touching outlines of the second layer.

IN is to extract SIN + TIN.

IN2, SIN2, TIN2 is to extract polygons of the second layer inside the first layer.

Note that input layer of **RELATE** must be merged polygon (OR1 or ROR*i*) for proper results. Output polygon is not merged polygon if input polygons of RELATE are not merged.

For example, in the next diagram, SIN is {1}, TIN is {2}, IN is {1, 2}.



- **OUT** - extract polygons of one layer outside the other layer

Syntax: [S/T]OUT[2] TempLayer

TempLayer is resulting layer of **RELATE** command.

There are three kinds of outside: SOUT (strict outside), OUTT (touch outside), OUT (outside).

SOUT is to extract polygons of the first layer defined by **RELATE** which are strict outside (no touching) of the second layer defined by **RELATE**.

TOUT is to extract polygons of the first layer which are outside of the second layer but have edges touching outlines of the second layer.

OUT is to extract SOUT + TOUT.

OUT2, SOUT2, TOUT2 is to extract polygons of the second layer outside the first layer.

Note that input layer of **RELATE** must be merged polygon (OR1 or ROR*i*) for proper results. Output polygon is not merged polygon if input polygons of RELATE are not merged.

For example, in the above diagram, SOUT is {5}, TOUT is {4}, OUT is {4, 5}.

- **OVER** - extract overlapping polygons of one layer from the other layer

Syntax: OVER[2] TempLayer

TempLayer is resulting layer of **RELATE** command.

OVER is to extract polygons of the first layer defined by **RELATE** which are overlapping with the second layer defined by **RELATE**. This is same as NOT OUT. Note that TOUT is excluded.

OVER2 is to extract polygons of the second layer overlapping with the first layer.

Note that input layer of **RELATE** must be merged polygon (OR1 or ROR*i*) for proper results. Output polygon is not merged polygon if input polygons of RELATE are not merged.

For example, in the previous diagram, OVER is {1, 2, 3}.

- **NOT** - extract opposite result of non-boolean relational operation

Syntax: NOT Op TempLayer

TempLayer is resulting layer of **RELATE** command.

Op is relational operation (IN, OUT, OVER).

Note that input layer of **RELATE** must be merged polygon (OR1 or ROR*i*) for proper results. Output polygon is not merged polygon if input polygons of RELATE are not merged.

3.4.4 Dimensional Commands

Dimensional commands needs **SetAreaMargin** to catch correct results on working area boundary. Safe margin is the maximum amount of all dimensional commands except **AREA** command. If no area margin is set, program will automatically calculate, report, and use that value.

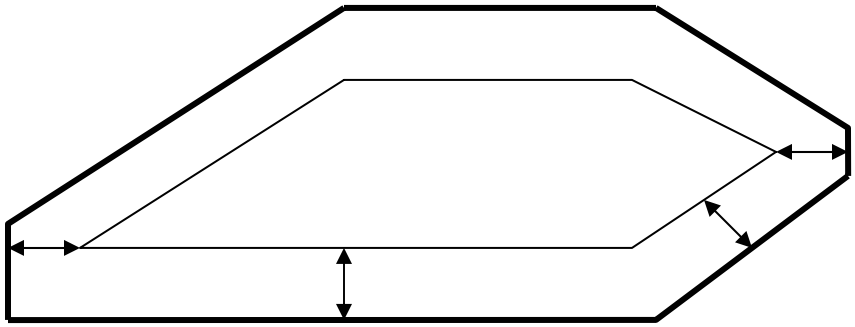
- **SIZING** - size up or down

Syntax: SIZING Layer Amount

Layer is { layer datatype } and Amount is by *nm*.

Size up (Amount > 0) or size down (Amount < 0), store in temporary layer, and return that temporary layer.

Note that we do smart customized sizing with horizontal and vertical cut-out for sharp angles. It is not either rectangle sweeping nor octagon sweeping sizing.



- **SPACE** - normal space checking

Syntax: SPACE Layer1 [Layer2] Amount [Option]

Layer1/Layer2 is { layer datatype }, Amount is by *nm*, and Option is "**corner**", "**edge**", or "**parallel**". Layers contain edges or polygons.

Produce polygons on Euclidean spacing violated locations, store in temporary layer, and return that temporary layer.

It applies exact corner-to-edge or corner-to-corner spacing. No corner-to-corner checking if input layers have edges.

If Layer2 is specified, it looks for spacing violation between two layers and spacing between the same layer is ignored. If one is inside of the other, it won't be considered as spacing violation. Only the spacing between outer edges of different layer will be considered.

If Option is "**corner**", it only checks corner-to-corner violations. If Option is "**edge**", it only checks edge-to-edge violations. If Option is "**parallel**", it only checks parallel edge-to-edge violations. If no Option is given, it checks all violations.

- **WIDTH** - normal width checking

Syntax: WIDTH Layer Amount [Option]

Layer is { layer datatype }, Amount is by *nm,* and Option is "**corner**", "**edge**", or "**parallel**". Layers contain edges or polygons.

Produce polygons on Euclidean width violated locations, store in temporary layer, and return that temporary layer.

It applies exact corner-to-edge or corner-to-corner width. No corner-to-corner checking if input layer has edges.

If Option is "**corner**", it only checks corner-to-corner violations. If Option is "**edge**", it only checks edge-to-edge violations. If Option is "**parallel**", it only checks parallel edge-to-edge violations. If no Option is given, it checks all violations.

- **ENC** - normal enclosure checking

Syntax: ENC Layer1 Layer2 Amount [Option]

Layer is { layer datatype }, Amount is by *nm,* and Option is "**corner**", "**edge**", or "**parallel**". Layers contain edges or polygons.

Polygons will be produced if Layer1 is not enclosed by Layer2 with specified Euclidean distance.

Produce polygons on enclosure violated locations, store in temporary layer, and return that temporary layer.

It applies exact corner-to-edge or corner-to-corner spacing. No corner-to-corner checking if input layers have edges (same as with **edge** option).

If Option is "**corner**", it only checks corner-to-corner violations. If Option is "**edge**", it only checks edge-to-edge violations. If Option is "**parallel**", it only checks parallel edge-to-edge violations. If no Option is given, it checks all violations.

4 Hierarchical Handling

You need Hierarchy license to run hierarchical operations except **HierarchyAnalysis**.

If user wants to convert/create layers, hierarchical handling is most desirable for later processes because once flattened, all the later processes must be done in flat.

This system adapts cell based pattern matching algorithm for hierarchical operations such that identified cell pattern must have the same target (intended) layers and environment layers within the specified halo (distance) from target layers of original cell. Output hierarchy will be same as original in terms of hierarchical tree structure of instances (i.e., global view) but it will be different in terms of cells. Newly created/copied cells (cell patterns) having different environments will have suffix of "_ic" or "_ich", and the number after those suffix. Basically, outputs of hierarchical operations will match those of flat operations not only for error detection but also for conversion.

Note that repeated hierarchical handling will result in exponential growth of data volume and run time (e.g., running additional hierarchical operations on output of previous hierarchical operations). It could be desirable to run hierarchical operations at one step as much as possible.

- **HierarchyAnalysis** - apply hierarchy analyzer and report

Syntax: HierarchyAnalysis

It will simply analyze hierarchy structure and report total compress ratio comparing to flattening simulation. You can determine if hierarchical handling is better or not based on such reported numbers.

This operation does not need Hierarchy license.

- **Hierarchical** - apply command set hierarchically

Syntax: Hierarchical { { command1 ... } { command2 ... } ... } or

Hierarchical { { source commands1.tcl } { source commands2.tcl } … }

Hierarchical handling is analogous to flat handling of **ForEachArea**.

It runs set of commands hierarchically at one step (only one time cell pattern matching is performed) based on **SetAreaMargin** (Halo), **Layers** (Target Layers), and **EnvLayers** (Environment Layers) parameters.

Note that providing accurate target/environment layers and halo affects performance and quality significantly. If user wants to run different target layers, it is desirable to run separate script. It is because hierarchy handling in this system is optimized for conversion, not for mere checking.

In most of cases, target layer is one layer while there can be several environment layers. It is strongly recommended to choose the least area layer as target layer when you have to choose one of two. E.g., between poly and diffusion, choose poly as target layer. Between via and metal, choose via as target layer. You might experience significant difference depends on your choice.

Also, if input data has very poor compress ratio comparing to flattened data (say, 2), run time and output might be worse than those of flat handling. Note that compress ratio is computed and reported in the beginning of hierarchical operation in this system. The initial compress ratio before hierarchical handling and post compress ratio after hierarchical handling are reported and user can decide if flat handling is better based on such reported numbers.

- **EnvLayers** – specifies environment layers for cell based pattern matching

Syntax: EnvLayers { { layer datatype } ... }

If empty (default), all existing layers will be considered as environment layers.

- **SetFlattenSize** – flatten cells if x or y size is smaller than the specified size, default is 0 and input unit is *nm*.

5 Sample Scripts

- **Boolean**

```
LogFile a.log
Library a.db
GDSIn a.gds
Areas { { "" 0 0 300 300 } }
ForEachArea {
      { TempLayer poly { OR {1 0} {1 1}} }
      { SaveDB poly {101 0} }
      {# The above is same as: SaveDB { OR {1 0} {1 1} } {100 0} }
      { TempLayer gate_rel { RELATE poly {10 0} } }
      { SaveDB {RAND gate_rel} {102 0} }
      { SaveDB {RDIFF gate_rel} {103 0} }
}
```

- **DRC**

```
LogFile drc.log; #log file
Library lib/a.db
OASISIn a.oas
Areas {{"" 0 0 0 0}}
ForEachArea {
      { TempLayer poly {OR1 {20 0}} }
      { SavePoly {SPACE poly 100} space.err }
      { SavePoly {WIDTH poly 100} width.err }
}
```

- **Hierarchical DRC**

```
LogFile ha.log; #log file
Library lib/a.db
Layers { {1 0} }
GDSIn gds/a.gds
#Areas { { "" 0 0 0 0 } }
#SetAreaMargin 240
Hierarchical {
      { TempLayer poly {OR1 {1 0}} }
      { SaveDB { WIDTH poly 240 }  {101 0} }
      { SaveDB { SPACE poly 240 }  {102 0} }
}
```

- **Multi Library Usage (XOR)**

```
LogFile xor2.log; # log file
Library lib/asizeup.db; # fist library by default
Layers {{100 0}}; #read {100 0} only
GDSIn gds/asizeup.gds
SetLibrary 2; # set to 2nd library for coming commands
```

```
Library lib/asizeup2.db
GDSIn gds/asizeup2.gds; #it will read {100 0} only too
#Areas { { "" 0 0 0 0 } }; #default don't need to specify
ForEachArea {
      {# get {100 0} from 1st library}
      {SetLibrary 1}
      {TempLayer L1 {OR1 {100 0}}}
      {# get {100 0} from 2nd lib, compare, and save xor result}
      {SetLibrary 2}
      {SavePoly {XOR L1 {100 0}} xor.err}
}
```

6 Commands Index