All About Calma's GDSII Stream Format

Steve DiBartolomeo
Applications Manager

Introduction

For over 25 years GDSII has been the industry standard database for IC layout. While other formats have been proposed to replace it (and one, OASIS, seems to be gaining some traction) GDSII remains by far the main way of describing the physical layout for the masks used to build a chip.

This is despite the fact that GDSII is not an open industry standard -- it was developed by Calma in the 80's and the ownership of the specification moved from Calma to GE to Valid to Cadence over the years. If you ask the right person you might even be able to get a copy of the GDSII specification from Cadence, though I have been working with a copy of the spec that I Xeroxed® back in 1989.

Improvements and Enhancements since 1989?

To my knowledge there has been no official updating of the GDSII specification since 1989. As you can imagine, computers and processors have come a long way since that time; the specification has some constraints which were probably based both on the limitations of the 80's era computers and also on the developer's beliefs that chips were going to be approximately the same complexity for the foreseeable future.

Fortunately, the way the database was created enabled users to extend many of the *dejure* limitations while maintaining *defacto* compatibility with the actual architectural underpinnings.

Simplicity

To some extent, GDSII's long life is due to both its elegant architecture and simplicity. The elegant architecture enables it to support today's modern chips with their billions of polygons while the simplicity enabled programmers to write code to manipulate GDSII and do things with it that the developers could not have imagined. This enormous base of legacy code is probably what is slowing the transition to alternatives (in particular OASIS.)

Basic Properties of GDSII

If you had to describe GDSII in only a few words these might be good ones to use:

Integer Database

Hierarchical

Binary

Record Based

Integer Database

GDSII is an integer database. The basic unit of measurement is a nanometer (10^{-9} meter)

Since four byte signed integers are used to describe a coordinate then the integer coordinates can range from from minus 2^{31} to plus 2^{31}-1. (two's complement)

Hierarchical

GDSII is organized in a hierarchical fashion. That is to say, that a number of elements are grouped into a cell or structure, and then that structure is referenced (or instanced or placed) multiple times. Since digital IC's are extremely repetitive, the database matches the physical layout approach very efficiently.

Cells can be nested with no limitation as to how deep the nesting goes (though I have yet to see nesting more than 9 levels deep.)

It is this nesting and hierarchy that allow one to describe an IC with one billion polygons using a database on the order of 5 GBytes ...

Unfortunately, when one needs to compute the actual position of the polygonal entities, one must "reverse" this nesting; for large databases this turns out to be a difficult computation to do quickly.

Binary

The database is binary for compactness. This means that any software for reading or writing GDSII has to be able to extract each byte and interpret the bits.

There is no official ASCII equivalent to the binary format. Various companies (including Artwork) have developed their own gdsii binary-ascii converters for those who wish to use tools such as Perl, awk or Python to manipulate GDSII data.

Record Based

GDSII is divided into "records." Only a few record types make up the great majority of the GDSII data. I've listed the essential ones here:

Records used once at the beginning of a GDSII file. The record number is shown in brackets.

```
BGNLIB[1]
LIBNAME[2]
UNITS[3]
ENDLIB[4]
```

Records used to begin/end a structure

```
BGNSTR[5]
STRNAME[6]
ENDSTR[7]
```

Records of Entities Within a Structure

```
BOUNDARY[8] [LAYER [13], DATATYPE [14], XY[16]]
```

```
PATH[9] [ LAYER[13], DATATYPE[14], PATHTYPE[33], WIDTH[15], XY[16] ]
TEXT[12][LAYER[13], DATATYPE[14], TEXTTYPE[22], PRESENTATION[23], STRING[25],
XY[16] ]
```

Records of References Within of a Structure

```
SREF [10] [ STRANS[26], MAG[27], ANGLE[28], XY[16] ]
AREF [11] [ STRANS[26], MAG[27], ANGLE[27], COLROW[19], XY[16]]
```

Page **1** | 2 | 3 | 4 | 5 | 6

ARTWORK CONVERSION SOFTWARE, INC.
417 Ingalls St. Unit C, Santa Cruz, CA 95060 831.426.6163 email: info@artwork.com

All About Calma's GDSII Stream Format [2]

Steve DiBartolomeo
Applications Manager

Hierarchy

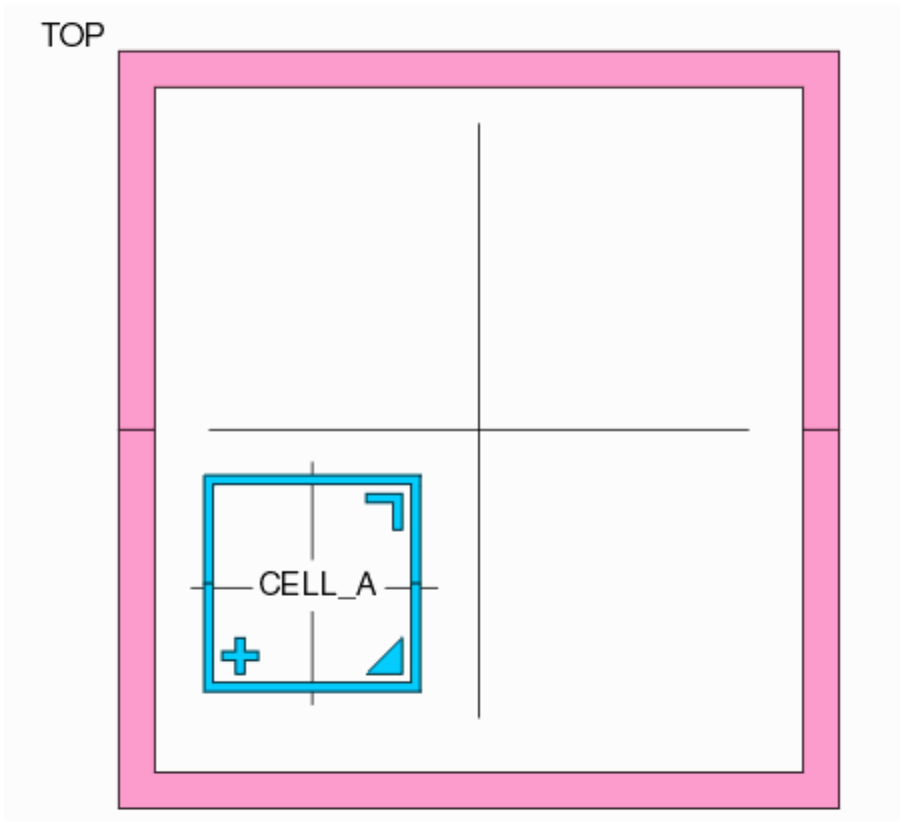
GDSII data is hierarchical which allows it to describe millions or billions of polygons with a much smaller number of database records.

The basic container is called a structure (often referred to as a cell). The structure can contain geometric elements such as boundaries and paths and references to other structures (SREFs and AREFs.) Let's look at a very simple example to understand what is going on.

```
BGNLIB                           start of the so called "library"

BGNSTR                           begin a structure
NAME=TOP                         the structure's name
BOUNDARY                         a geometric boundary
BOUNDARY                         a geometric boundary
SREF CELL_A [X,Y,ROT,MIR,MAG]    a reference placing another structure
ENDSTR                           end the structure

BGNSTR                           begin a new structure
NAME=CELL_A                      the structure's name
BOUNDARY                         a geometric boundary
BOUNDARY                         a geometric boundary
BOUNDARY                         a geometric boundary
BOUNDARY                         a geometric boundary
BOUNDARY                         a geometric boundary
ENDSTR                           end the structure

ENDLIB                           end the library
```

In the above pseudo GDSII code we have two structures defined: TOP and CELL_A. TOP contains a reference to CELL_A, essentially placing it somwhere with some transformations. CELL_A is very simple one and has no references to other cells (though it could.)

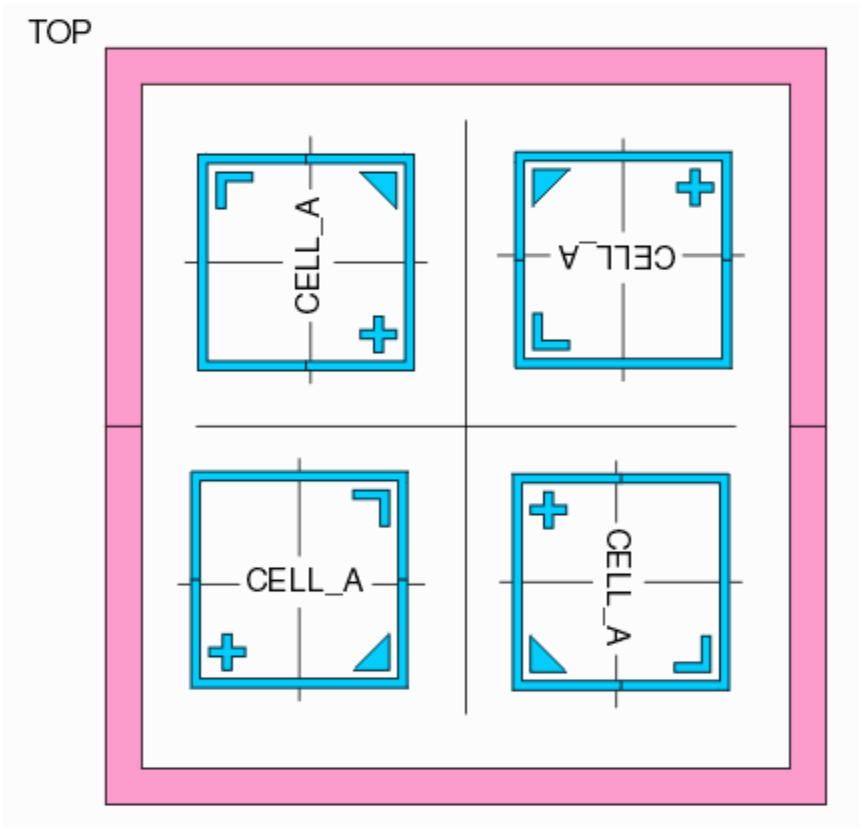
Here is a picture assuming that CELL_A (the smaller blue one) is placed in the lower left quadrant relative to the center of TOP.



Now TOP could easily reference CELL_A 4 times - each time in a different position and rotation.

```
BGNLIB                               start of the so called "library"

BGNSTR                               begin a structure
NAME=TOP                             the structure's name
BOUNDARY                             a geometric boundary
BOUNDARY                             a geometric boundary
SREF CELL_A [X1,Y1,ROT1,MIR,MAG]     a reference placing another structure
SREF CELL_A [X2,Y2,ROT2,MIR,MAG]     a reference placing another structure
SREF CELL_A [X3,Y3,ROT3,MIR,MAG]     a reference placing another structure
SREF CELL_A [X4,Y4,ROT4,MIR,MAG]     a reference placing another structure
ENDSTR                               end the structure
```

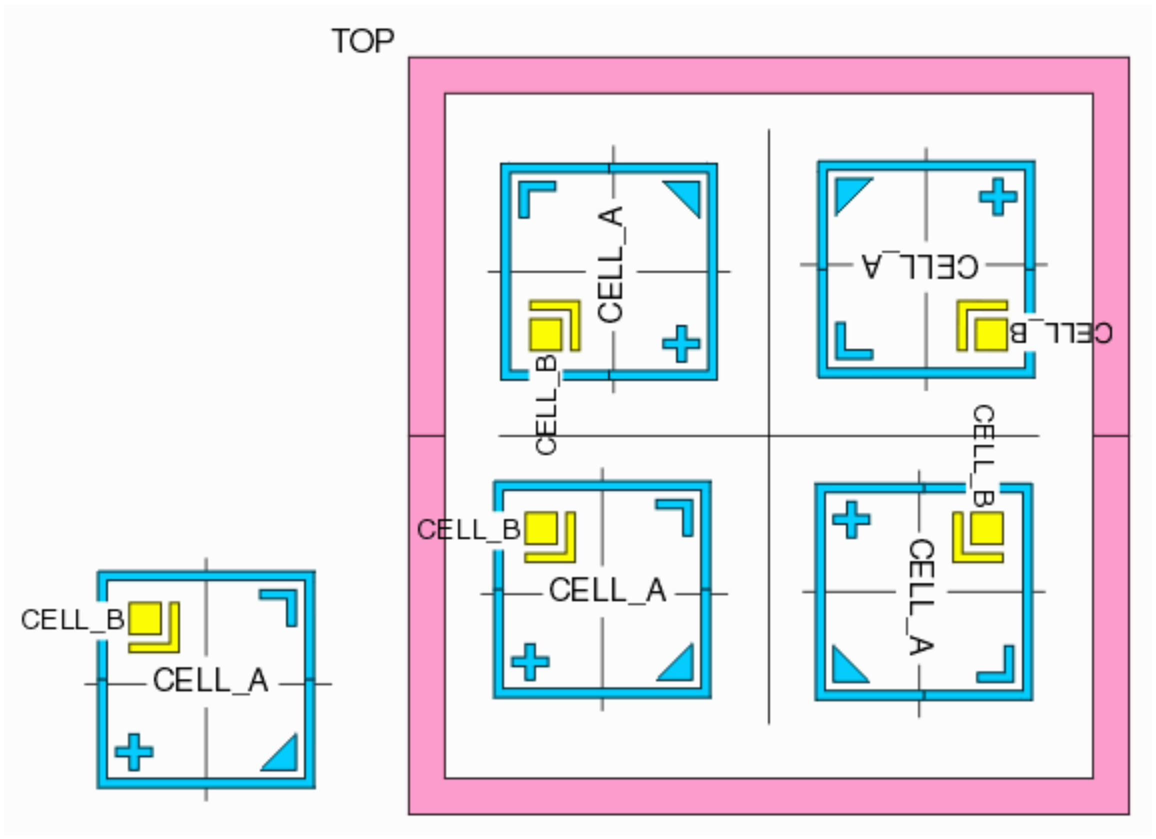
Here is what it might look like:



Additional Levels of Nesting

CELL_A could reference another cell - CELL_B. [Note that it is forbidden for CELL_A to reference a cell that references it; this would create an impossible recursion.] If that was the case, each time TOP referenced CELL_A, CELL_B would also end up generated.

```
BGNSTR                                 begin a structure
NAME=TOP                               the structure's name
BOUNDARY                               a geometric boundary
BOUNDARY                               a geometric boundary
SREF CELL_A [X1,Y1,ROT1,MIR,MAG]       a reference placing another structure
SREF CELL_A [X2,Y2,ROT2,MIR,MAG]       a reference placing another structure
SREF CELL_A [X3,Y3,ROT3,MIR,MAG]       a reference placing another structure
SREF CELL_A [X4,Y4,ROT4,MIR,MAG]       a reference placing another structure
ENDSTR                                 end the structure

BGNSTR                                 begin a new structure
NAME=CELL_A                            the structure's name
BOUNDARY                               a geometric boundary
BOUNDARY                               a geometric boundary
BOUNDARY                               a geometric boundary
BOUNDARY                               a geometric boundary
BOUNDARY                            a geometric boundary
SREF CELL_B [X,Y,ROT,MIR,MAG]       a reference placing another structure
ENDSTR                              end the structure

BGNSTR                              begin a new structure
NAME=CELL_B                         the structure's name
BOUNDARY                            a geometric boundary
BOUNDARY                            a geometric boundary
ENDSTR                              end the structure
```



Order and Sorting?

A programmer would note that there ought to order in the database - i.e. one would prefer that a cell be defined before it is referenced. It turns out that GDSII has no such requirement and one will regularly encounter files where structures are referenced early on and only defined later.

This makes rendering the layout more work since one cannot make a single pass through the database and be able to compute what the image will look like.

For small layout files the extra pass is not a big deal but as the file size grows (and it does, to many gigabytes) this lack of "order" becomes a significant penalty when "opening" the data.

Optimizing the initial opening of the file and creating a way to quickly access elements in any particular geometric window turns out to be both very important in dealing with large files and devilishly difficult to do efficiently. This is what separates the men-from-the-boys when it comes to code for viewing, plotting and manipulating GDSII files.

Page 1 | **2** | 3 | 4 | 5 | 6

All About Calma's GDSII Stream Format [3]

Steve DiBartolomeo
Applications Manager

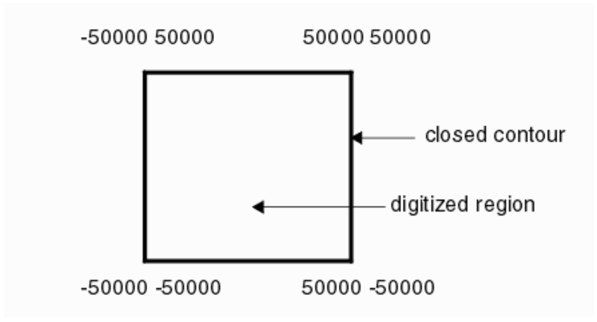
GDSII BOUNDARY

The GDSII BOUNDARY record (geometric entity) is one of the two key geometries used to describe the layout (the other being the PATH)

The boundary consists of a list of vertices which are connected by straight edges to form a closed contour. The "inside" of the contour is called the digitized area.

```
BOUNDARY 5 0  <-- layer 5,
datatype 0
-50000 -50000 <-- First X,Y
vertex
50000 -50000  <-- Second X,Y
vertex
50000 50000   <-- Third X,Y
vertex
-50000 50000  <-- Fourth X,Y
vertex
-50000 -50000 <-- Back to First
X,Y vertex
ENDEL
```

[Note: Since this example assumes units of UM and a resolution of 1000 UM per GDSII db tick, each integer value is 1000x the value in UM]



Number of Vertices per Boundary

The GDSII specification limits the number of vertices to 200.

This limit is totally arbitrary and was probably set at the time to limit the amount of computing power needed to render, store and process polygons. Most modern GDSII readers and writers support a much larger number such as 1024, 2048, or 4096. The actual upper limit is set by the design of the database. Since one of the records in the GDSII boundary defines the number of vertex pairs to follow, and that record is two bytes long the maximum number of coordinates would be 8191.

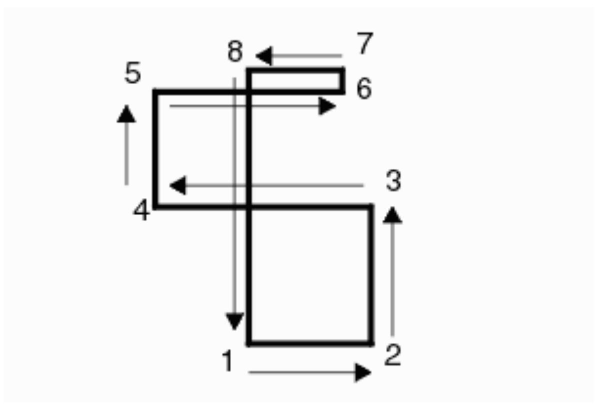
If you are developing a GDSII reader you should use the 8191 number as the maximum number of vertex pairs to ever expect. If you are writing a GDSII reader you should make the max number of vertices per polygon configurable so that the user can produce a GDSII file compatible with the target that is to read it.

Direction?

The direction of the vertices (i.e. clockwise or counterclockwise) has no meaning in GDSII and polygons can be written without concern as to their direction of their vertex order.

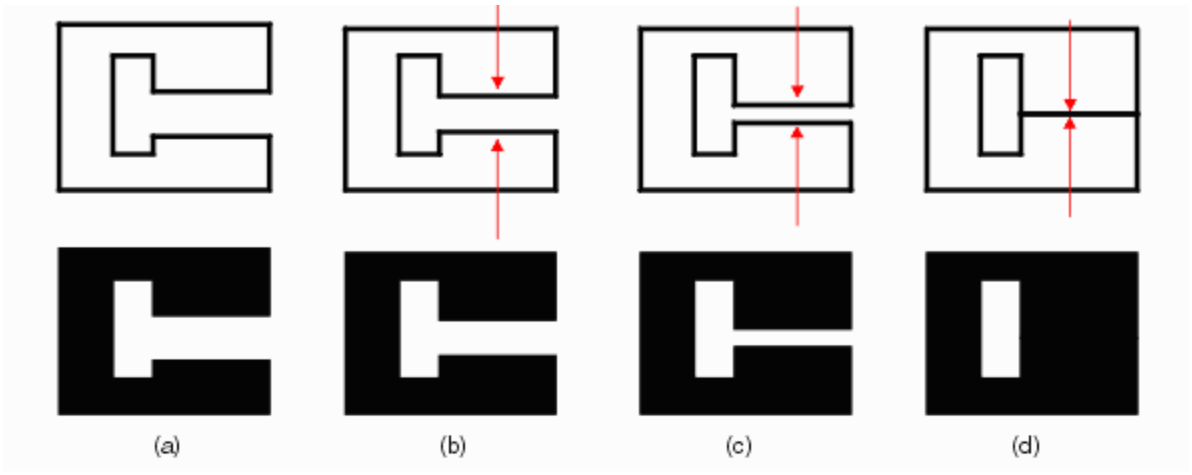
Self Crossing

The contour forming a boundary may not cross over on itself. This is considered an illegal behavior though some CAD systems will process such a polygon. However when developing a GDSII writer, polygons should be checked for self intersection prior to writing it out.



Self Touching

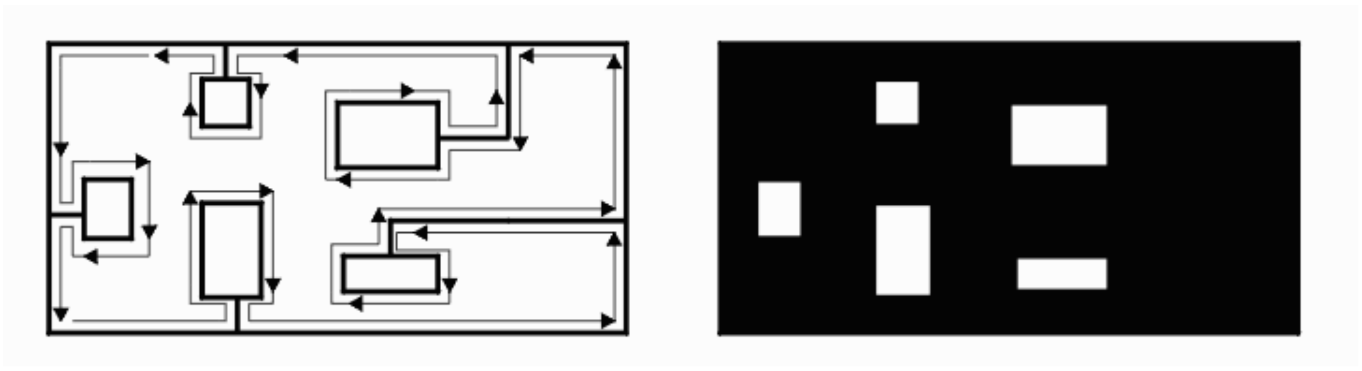
Self touching (also known as "re-entrant" or "keyhole") is not quite the same as self intersection and is actually allowed in GDSII. However it has to be done "correctly." Self touching is generally used to form islands or holes inside of a digitized region. The series of images below shows how a self touching, also known as re-entrant, boundary is created and how it will look on a mask when processed.



Polygon (a) is clearly legal as it does not self intersect or self-touch. Now imagine that the facing edges are slowly pushed towards each other (b) and (c) until they finally meet (d). Is (d) a legal polygon? Yes. It is not considered to be self intersecting but it does touch itself.

Multiple Holes

Multiple "holes" in a single boundary are common as shown below:



Interaction Between Boundaries

Each boundary stands alone and from a database point of view there is no interaction between boundaries. Other CAD systems may treat an "inner" boundary as a hole or cutout but this is not the case with GDSII.

Manhattan Data

Often one will hear the term "Manhattan." This means that all of the geometric data edges run vertical or horizontal. GDSII data need not be Manhattan but, mostly for computational reasons most digital IC layouts still used purely Manhattan data. Analog, high voltage and RF integrated circuits often use non-manhattan (also known as all-angle) layout.

Page 1 | 2 | **3** | 4 | 5 | 6

ARTWORK CONVERSION SOFTWARE, INC.

417 Ingalls St. Unit C, Santa Cruz, CA 95060 831.426.6163 email: info@artwork.com

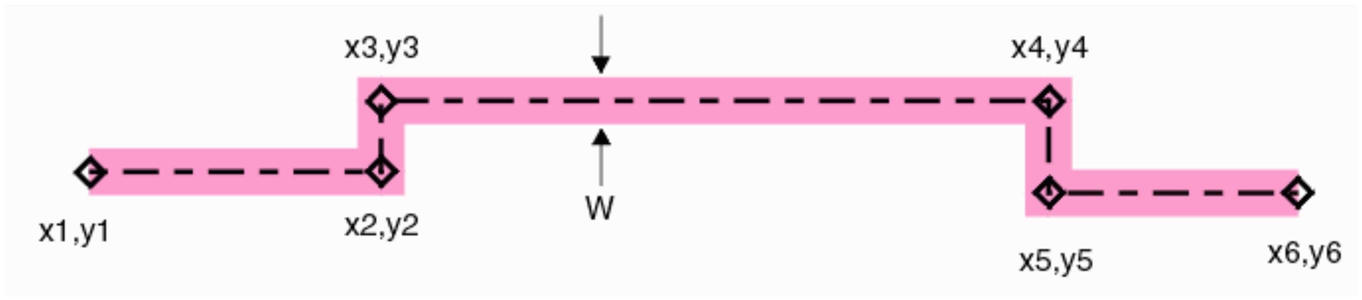
All About Calma's GDSII Stream Format [4]

Steve DiBartolomeo
Applications Manager

Paths

The PATH is one of the two base geometric entities used to describe the layout (the other being the BOUNDARY)

The PATH consists of two or more vertices that describe the center point of a line. The path normally has a finite width. This is a straightforward concept except for the bends in the path and the start/end of the path.



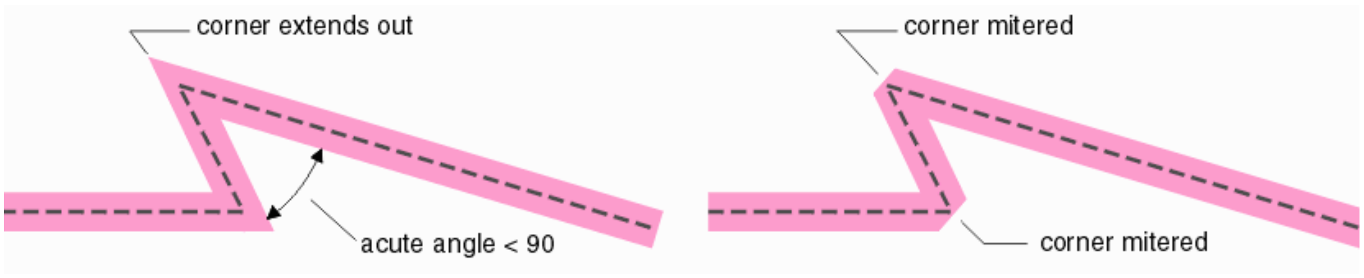
Number of Vertices per Path

The GDSII specification limits the number of vertices to 200.

Most GDSII readers can support a much larger number of vertices. The maximum is 8191 vertices based on the two bytes available in the path record that describes the number of XY records to follow.

Bends

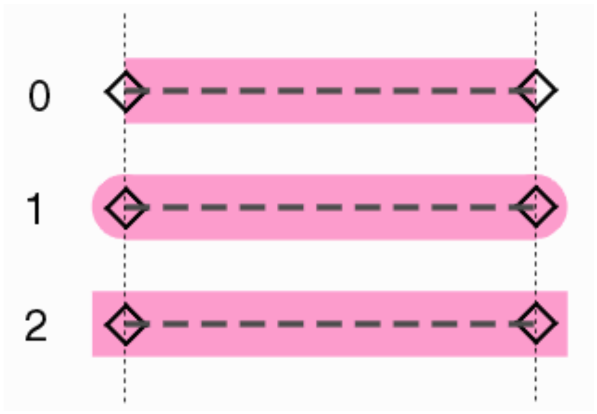
The GDSII specification says nothing about how the bends in a path should be rendered. For 90 degree bends there is no confusion. However for bends with acute angles most rendering algorithms assume some amount of mitering - the exact formula for the miter will vary from implementation to implementation.



End Caps

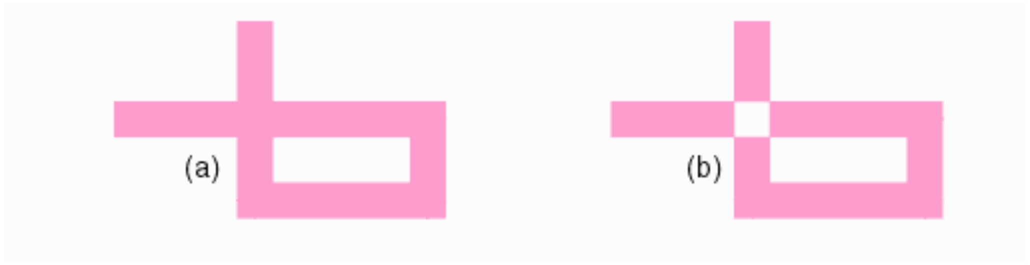
The path record has a flag that indicates the nature of the start and end (known as the end caps.) There are three standard values and a fourth flag that indicates a custom value.

```
Flush
1=Half Round Extension
2=Half Width Extension
4=Custom Extension
```



Self Intersecting

Self intersecting paths (a) should not be written out to GDSII. Some tools that read GDSII will correctly interpret them but many will generate unexpected results (b) or even crash.



Page 1 | 2 | 3 | **4** | 5 | 6

All About Calma's GDSII Stream Format [5]

Steve DiBartolomeo
Applications Manager

SREF - Structure Reference

We mentioned that GDSII is a hierarchical database. This is an essential feature - otherwise it would be impractical to use it to describe IC layouts that have millions or even billions of boundaries and paths. The hierarchy is valuable because IC designs tend to re-use the same geometric patterns over and over again.

The STRUCTURE record holds boundaries, paths, SREFs and AREFs. One can use an SREF (contained in a different structure) to reference or "place" a structure. In that fashion the complete list of boundaries, paths and references need not be explicitly enumerated over and over again.

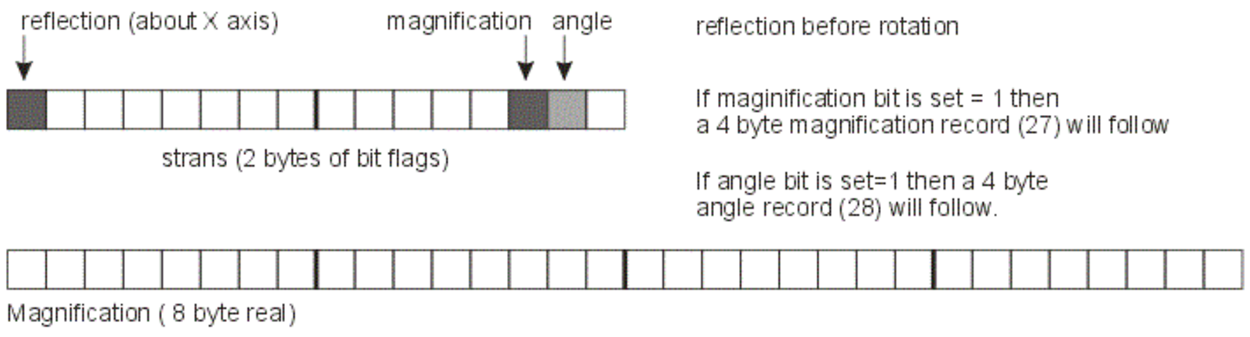
The SREF record looks like this:

```
SREF [ELFLAGS] SNAME  [<strans>]  XY
```

where:

```
SREF       - string identifying this as a SREF
ELFLAGS    - rarely used binary flags (see plex)
SNAME      - the name of the structure being referenced
strans     - two byte bit array controlling reflection, magnification and rotation.
XY         - placement coordinates of the instance
```

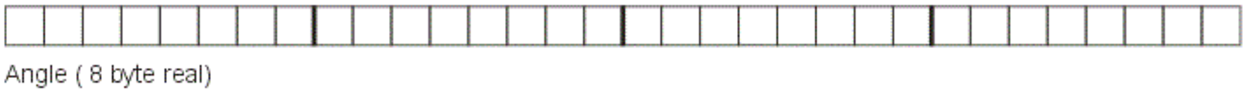
STRANS Details


reflection (about X axis)
magnification
angle
strans (2 bytes of bit flags)
Magnification (8 byte real)

reflection before rotation

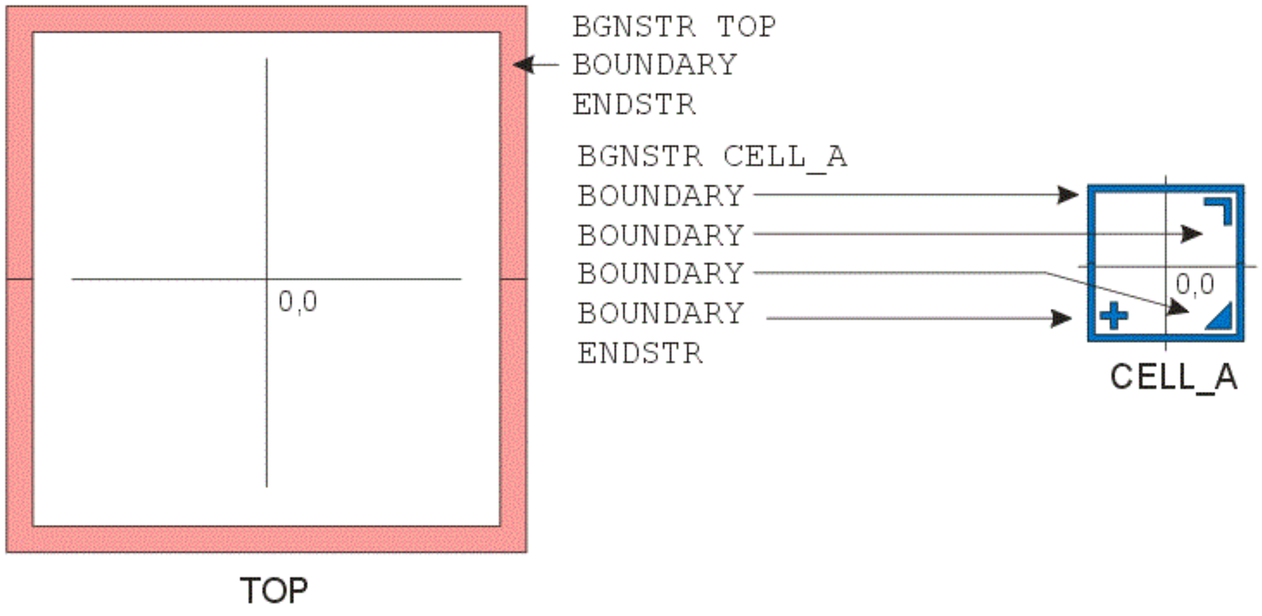
If maginification bit is set = 1 then
a 4 byte magnification record (27) will follow

If angle bit is set=1 then a 4 byte
angle record (28) will follow.

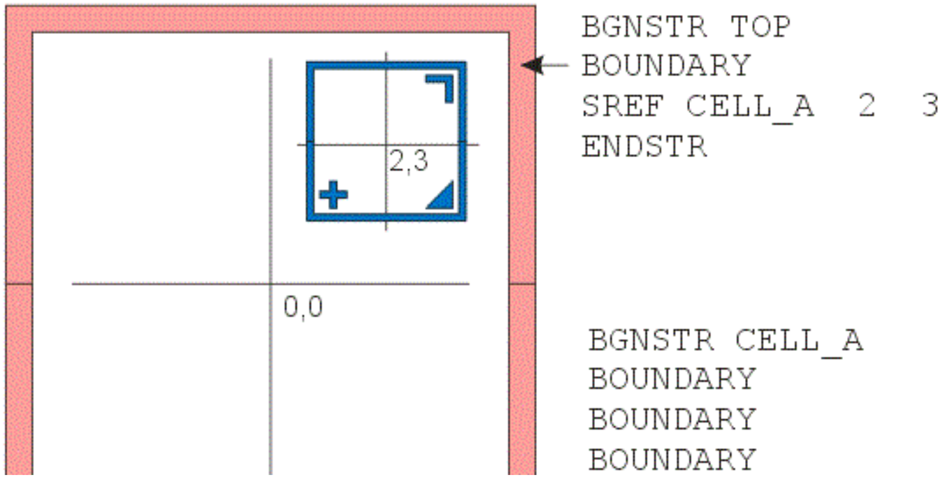


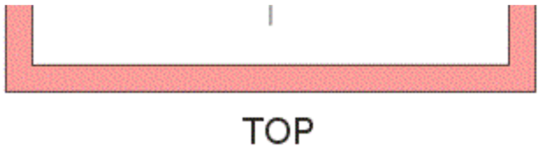
Examples of SREF and Transformations

Suppose we have created a GDSII file with two separate structures: TOP and CELL_A. Initially we have no SREFs in the file so the two structures have no relationship. This is shown diagrammatically below:



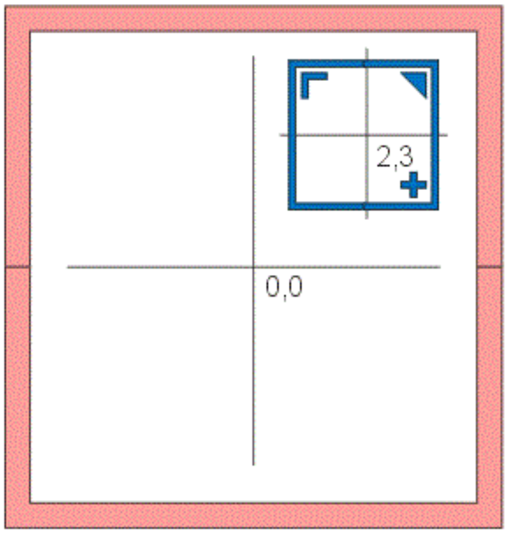
Now let's add a SREF record to TOP that points to CELL_A. We will only have a simple XY insertion at 2,3 with no transformations.





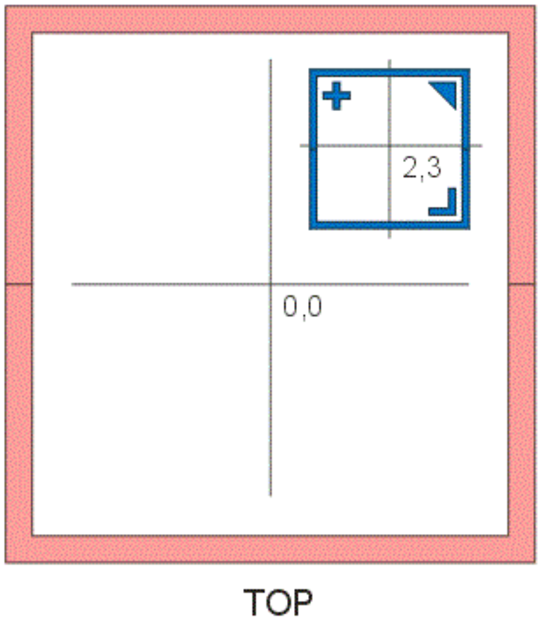
```
BOUNDARY
ENDSTR
```

This time let's place CELL_A at 2,3 but let's also rotate it 90 degrees counter clockwise (CCW).



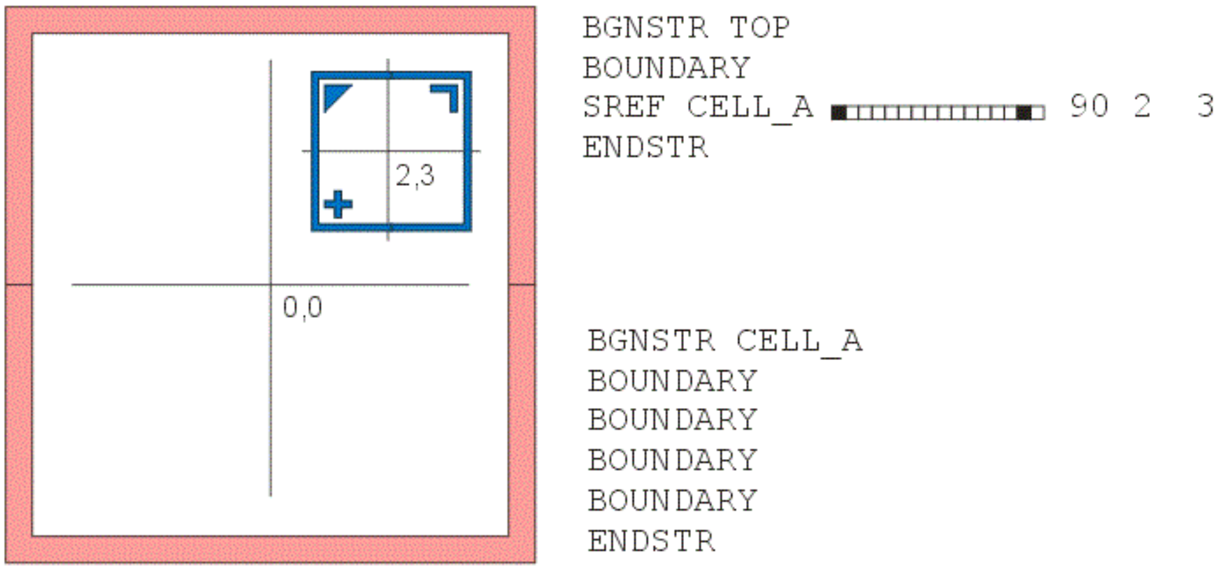
```
BGNSTR TOP
BOUNDARY
SREF CELL_A □□□□□□□□□□□□□□■□ 90 2  3
ENDSTR

BGNSTR CELL_A
BOUNDARY
BOUNDARY
BOUNDARY
BOUNDARY
ENDSTR
```

This time let's place CELL_A at 2,3 and let's turn on the reflection (about X) flag.



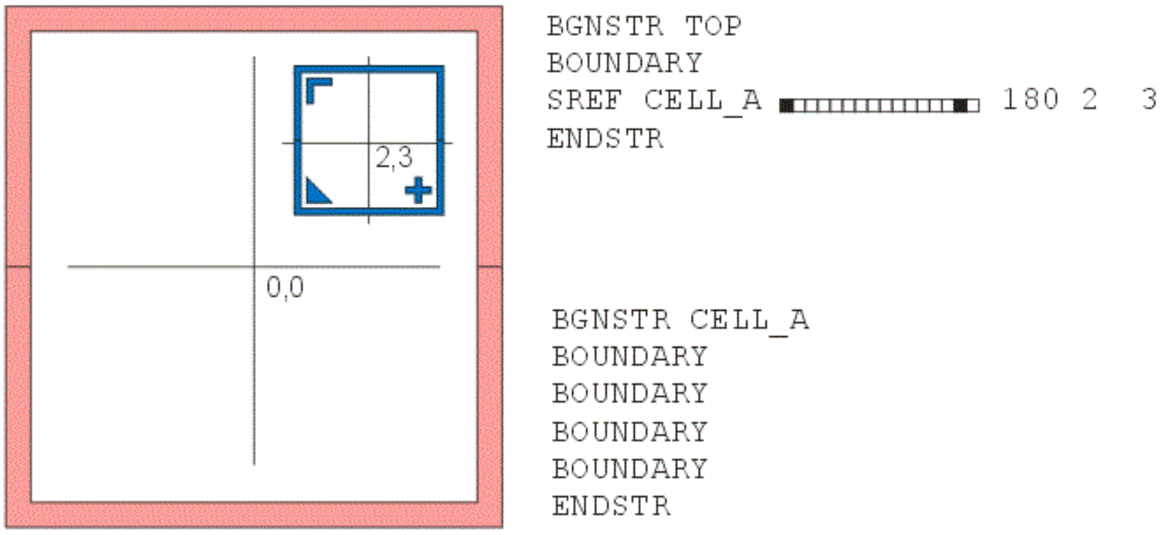
```
BGNSTR TOP
BOUNDARY
SREF CELL_A ■□□□□□□□□□□□□□□□  2  3
ENDSTR

BGNSTR CELL_A
BOUNDARY
BOUNDARY
BOUNDARY
BOUNDARY
ENDSTR
```

This time let's place CELL_A at 2,3 let's turn on the mirror flag and let's rotate 90 CCW.



```
BGNSTR TOP
BOUNDARY
SREF CELL_A ■□□□□□□□□□□□□□■□ 90 2  3
ENDSTR

BGNSTR CELL_A
BOUNDARY
BOUNDARY
BOUNDARY
BOUNDARY
ENDSTR
```

This time let's place CELL_A at 2,3 let's turn on the mirror flag and let's rotate 180 CCW.



```
BGNSTR TOP
BOUNDARY
SREF CELL_A ■□□□□□□□□□□□□□■□ 180 2  3
ENDSTR

BGNSTR CELL_A
BOUNDARY
BOUNDARY
BOUNDARY
BOUNDARY
ENDSTR
```

Notice that the combination of reflection about X combined with a 180 degree rotation is equivalent to a reflection around Y. This is useful since there is no Y reflection flag. Reflection is performed prior to rotation.

Page 1 | 2 | 3 | 4 | **5** | 6

ARTWORK CONVERSION SOFTWARE, INC.

417 Ingalls St. Unit C, Santa Cruz, CA 95060 831.426.6163 email: info@artwork.com

All About Calma's GDSII Stream Format [6]

Steve DiBartolomeo
Applications Manager

The Array Reference

In addition to the SREF (structure reference) which is a single placement of a structure (cell), GDSII has an AREF (array reference) which enables placement of 1D and 2D arrays. This is very useful for many types of IC layouts that include memory; identical memory cells are often arrayed in two dimensions.

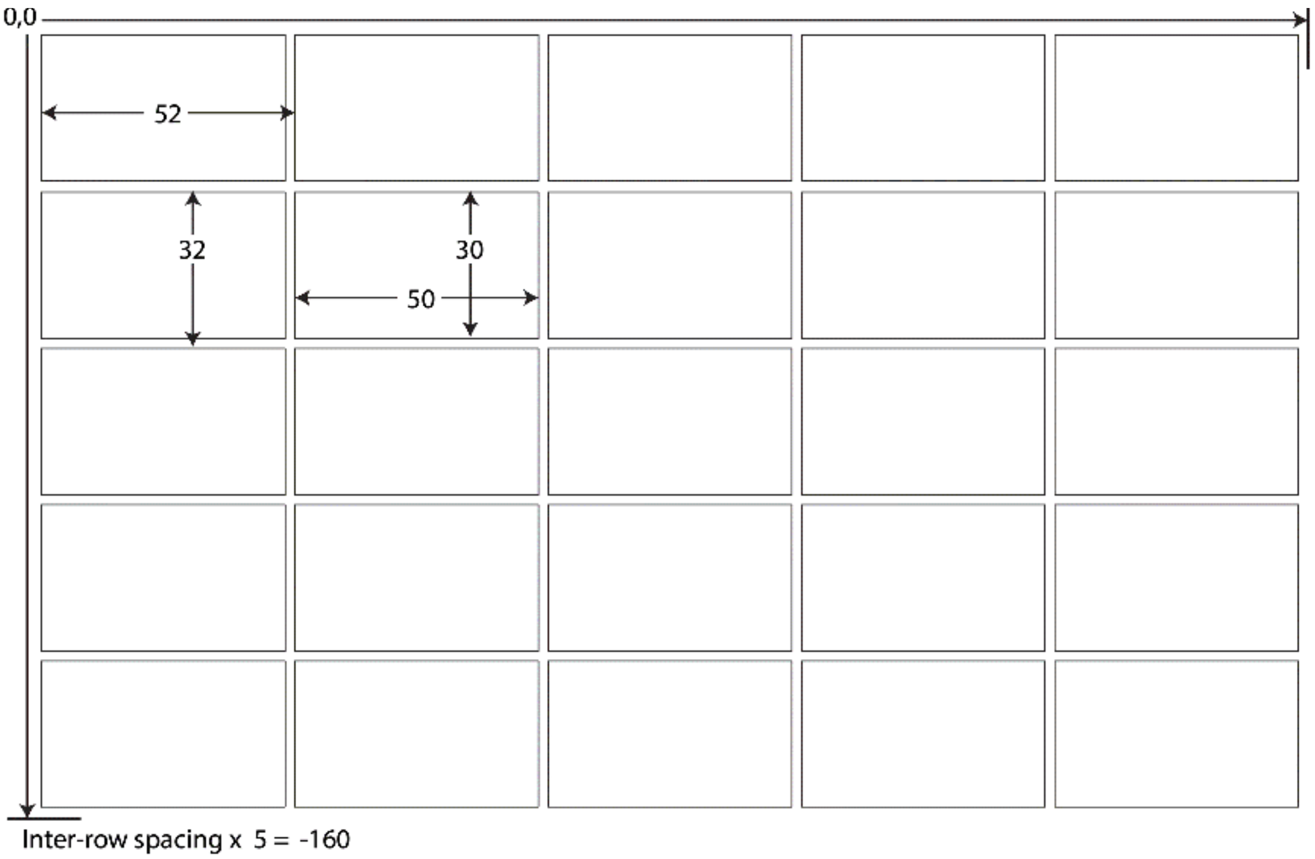
An AREF record has a number of arguments:

1. Name of Structure to Array
2. Mirror Flag (M0 or M1)
3. Scale Factor
4. Rotation
5. Number of Columns
6. Number of Rows

Then, there are three sets of coordinates that must be supplied:

7. the Array Reference Point
8. a position displaced from the Reference Point by the inter column spacing x the number of columns
9. a position displaced from the Reference Point by the inter row spacing x the number of rows

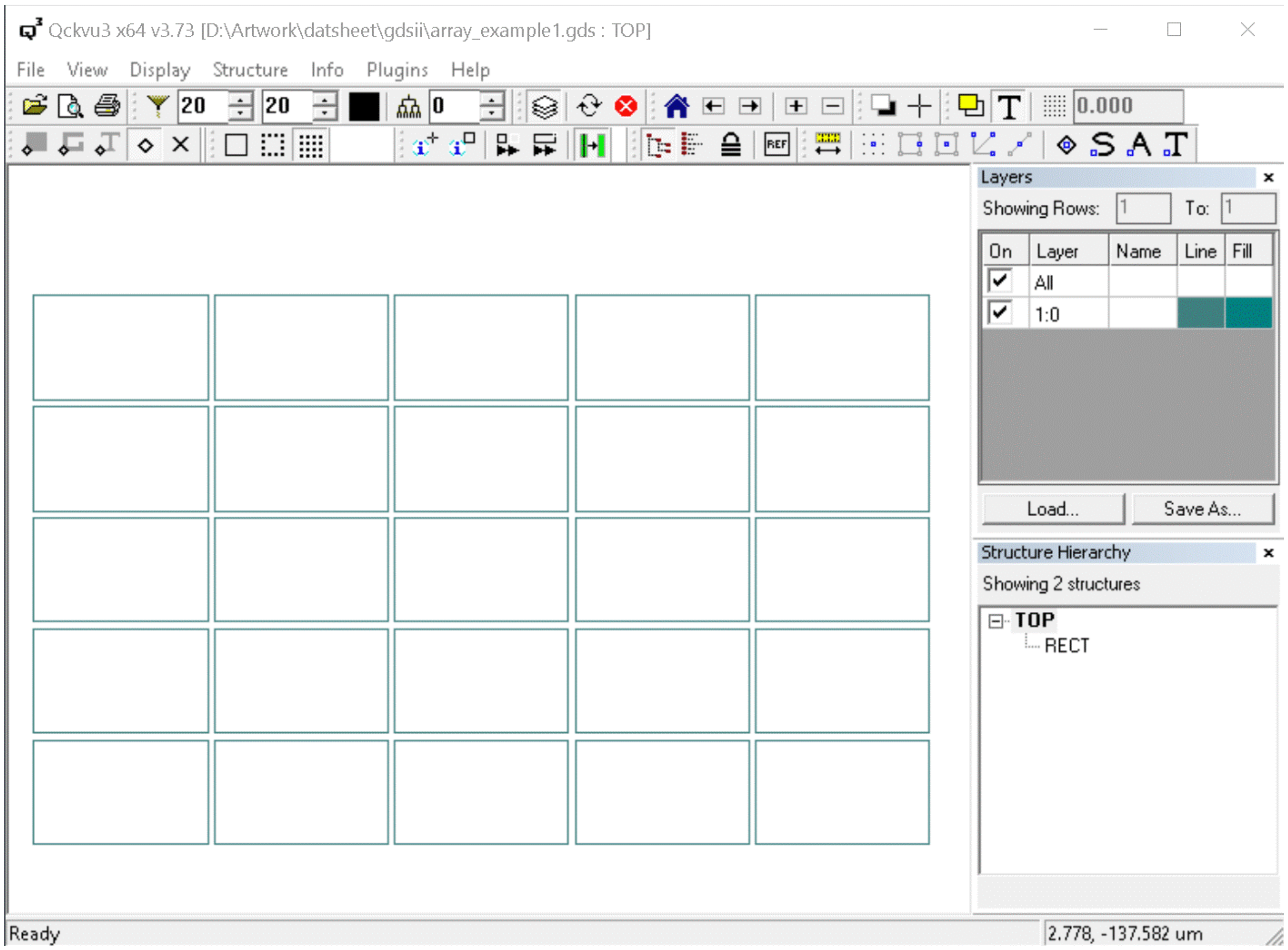
Let's look at a 5 x 5 array of cells that have an extents of 50 um x 30 um and a gap of 2 um between them.



We are going to write an ASCII "equivalent" of a small GDSII file using this array.

```
LIBRARY ARRAY_EXAMPLE1.GDS unit:UM grid:1000
STRUCT RECT
BOUNDARY 1 0
0 0
50000 0
50000 -30000
0 -30000
0 0
ENDEL
ENDSTR
STRUCT TOP
AREF RECT M0 1.0 0.0 5 5 0 0 260000 0 0 -160000
ENDEL
ENDSTR
ENDLIB
```

If we use Artwork's ascii2gds program to convert this to GDSII here is what we see:


Qckvu3 x64 v3.73 [D:\Artwork\datsheet\gdsii\array_example1.gds : TOP]
File View Display Structure Info Plugins Help
Layers
Showing Rows: 1 To: 1
Load... Save As...
Structure Hierarchy
Showing 2 structures
TOP
RECT
Ready
2.778, -137.582 um

On	Layer	Name	Line	Fill
✓	All			
✓	1:0		🟩	🟩

Rotating the Array

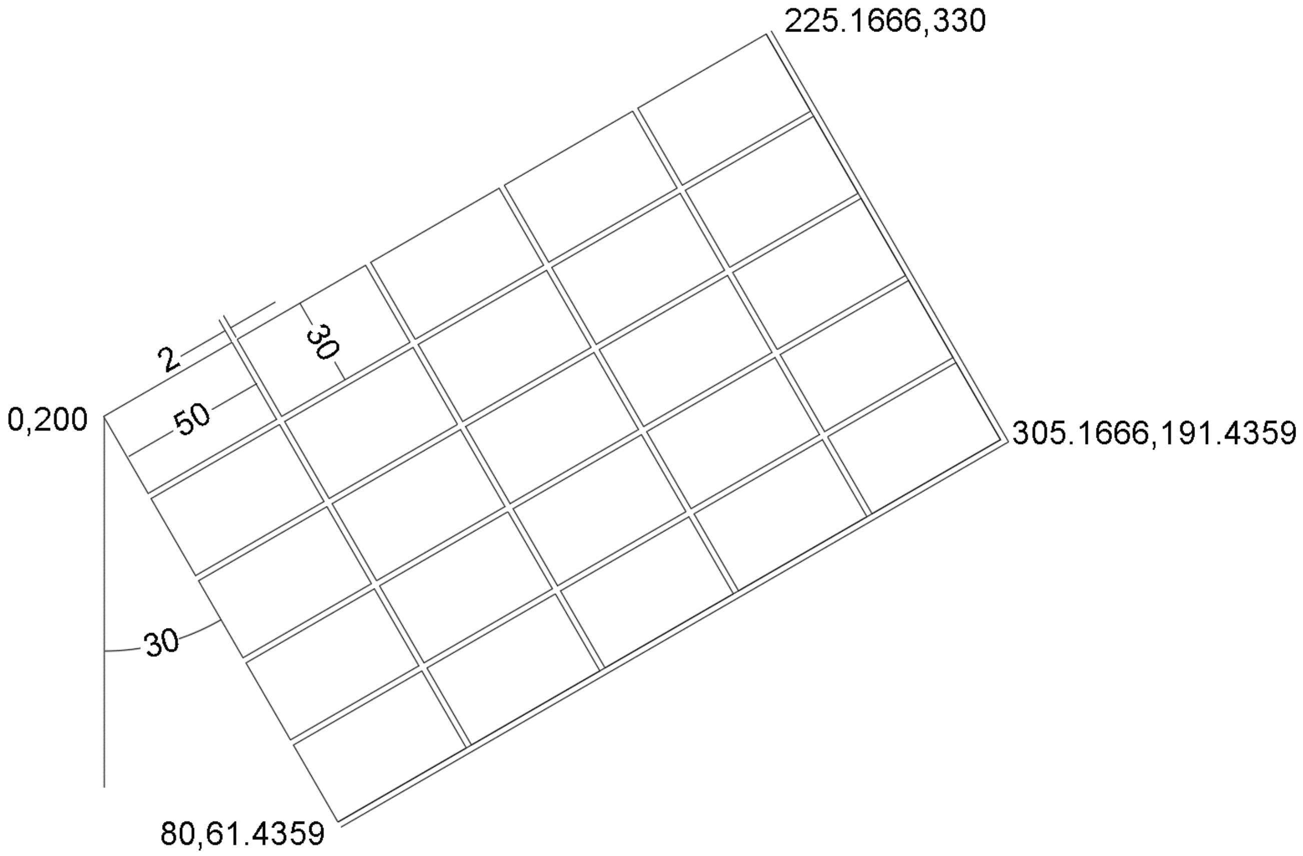
One would think that rotating this array by 30 degrees would be a simple matter of changing the rotation parameter, i.e.:

```
AREF RECT M0 1.0 30.0 5 5 0 0 260000 0 0 -160000
```

However this won't work! One actually has to compute the second and third pair of coordinates of the rotated array and enter those. I have no idea why this is the case but it is. So for the same array (but with reference point 0 20000 (instead of 0 0) we would have the following parameters in our AREF.

```
AREF RECT M0 1.0 30.0 5 5 0 200000 225167 330000 80000 61436
```

You can see in the illustration below how the locations for the second and third pair of coordinate parameters are calculated:



Page 1 | 2 | 3 | 4 | 5 | **6**

ARTWORK CONVERSION SOFTWARE, INC.

417 Ingalls St. Unit C, Santa Cruz, CA 95060 831.426.6163 email: info@artwork.com