Jens Lienig, Juergen Scheible
Fundamentals of Layout Design for Electronic Circuits
ISBN 978-3-030-39283-3, Springer 2020

3.2 Layout Data: Layers and Polygons

As indicated in Fig. 3.1, the layout data are the result of physical design. These data are used not only to store a design result and to prepare this result for the fab; rather, a layout engineer works continuously with the layout data throughout the design process. Hence, we next look more closely at the structure of this data and the key graphics operations that the layout engineer may apply to such data.

3.2.1 Structure of Layout Data

We have already explored several of the most important aspects of the layout data in Chap. 1 (Sect. 1.3). We have seen that chip layout data are comprised only of graphical data, and that these graphics contain all the information necessary to produce masks. The same is true for PCB layout data, where the layout is described by polygon coordinates, accompanied by data containing diameters and positions for drilling the via holes and data for positioning the devices.

In general, a graphic can be represented as a raster or vector image. While raster graphics are bitmaps, i.e., a grid of individual pixels that collectively compose an image, vector images are mathematical calculations from one point to another that form lines and shapes.

Electronic layout data are saved only as vector images and processed as such. There are several reasons for this: (i) the data structure of vector graphics suits layout representations ("polygons") very well; (ii) they do not need much memory as compared to raster images; (iii) they can be processed more quickly and easily due to the information they contain; and (iv) they can be reshaped without loss of accuracy. The only disadvantage of the vector data structure is that it must be converted to raster data for presentation on computer screens. This is not a problem nowadays as state-of-the-art design environments are readily available with very efficient algorithms and high-performance hardware.

Layers
Graphics elements in layout data, for example the lateral structure of a doped region or an interconnect, are called *shapes*. Each shape is assigned to a unique *layer*. This layer association is an elemental attribute of every shape that enables it to be assigned to masks. It also forms the basis for graphical linking operations, as we shall see.

There is one important point we would like to make clear at this stage. When we talk about "layers" in the context of layout data, we are referring only to the

above attribute in connection with the data structure. Such a layer often corresponds directly with a counterpart in the fabrication process. This counterpart could be a doped region in an "Nwell" layer, or "Metal1" for a metallic layer deposited on a wafer. This does not always have to be the case, however. Layout data also contain layers not directly associated with anything on a wafer, as we shall see (Sect. 3.3.4). The opposite can also be true: for example, the gate oxide layer on silicon is not modeled as a layer in the layout data.

We differentiate between the two types of "layers", as follows: we call layers used in the data structure as *drawn layers* and layers used in the fabrication process as *fabricated layers*. We shall only use this extended terminology where extra clarity is needed, i.e., in cases where the risk of misunderstanding is high if it is not used. In all other cases, and for the sake of simplicity, we will use the term "layers". Accordingly, all layers referred to in this chapter are "drawn layers" in the layout.

Shapes

The shapes in the layout data are always polygons. A polygon is a two-dimensional, continuous graphics element bounded by straight edges. This type of shape can be efficiently stored in the vector data structure as a list of successive corner coordinates. The resulting closed polyline determines the polygon; whether the polygon is to the left or right of the polyline must be defined, nevertheless. This varies from tool to tool. In some tools, the first coordinate in the list is appended at the end of the list, thus terminating the list.

Figure 3.6a shows an example of a general polygon with seven corners. Its coordinates are designated by C_i comprising two numerical values (x_i, y_i). By defining the smallest permissible grid (often called "manufacturing grid", "working grid", or simply "grid"), integer values, i.e., whole numbers, can be used for the (x_i, y_i) coordinates. Computer memory can thus be saved and the accuracy of the model is well-defined.

Donuts. Polygons with holes (also called *donuts*) can be modeled with this data structure. For efficiency in processing, this approach requires "dual" edge sequences in the data structure that "run" in two directions for this section. These superimposed edge pieces do not form a real polygon edge. An example is shown in Fig. 3.6b. The

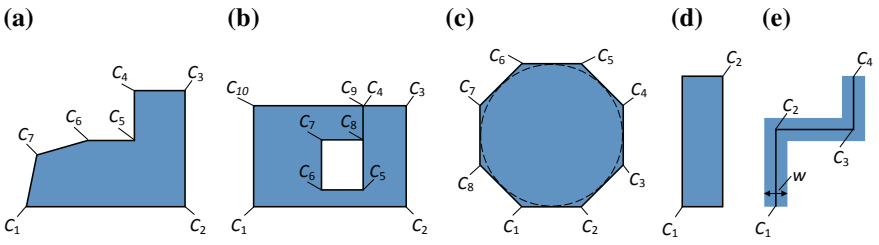

Fig. 3.6 Different shapes in a layout data structure, such as a general polygon (**a**), a polygon with a hole ("donut", **b**), a polygon that approximates round boundaries ("conics", **c**), a rectangle polygon (**d**) and a path polygon (**e**)

path (C_8–C_9) lies on the path (C_4–C_5) in this example. The coordinates C_4 and C_9 do not constitute real corners.

Conics. Graphics elements with round boundaries (some tool providers refer to these as *conics* in this context) cannot be precisely represented by vector images based on polygons. Round boundaries are always approximated by many straight-edge pieces; the level of approximation varies from tool to tool. For example, the number of edge pieces for a circle can be defined as a parameter. A circle approximated by a polygon with eight corners is shown in Fig. 3.6c.

Aside from general polygons, there are two other custom shapes: the *rectangle* and the so-called *path*. They are special cases of polygons, which can be more efficiently modeled in the data structure due to their special properties. Given that rectangles and paths are by far the most common graphics elements in a typical layout, this advantage is fully utilized in layout representations.

Rectangles. A rectangle is a polygon with four sides and four right angles. If the sides of the rectangle are parallel to the axes of the Cartesian coordinate system used for the design (which is almost always the case), a rectangle can be modeled with only the coordinates of two diagonally opposed corners (Fig. 3.6d). The data volume can thus be almost halved. This efficient data structure matches the way a rectangle is created in the graphics editor, that is, by the digitization of these two opposing-corner coordinates.

Paths. Paths are polylines to which a specific *width w* is assigned. These shapes are typically employed for the fabrication of interconnects to provide the electrical current with a continuous, constant interconnect cross-sectional area. The center line of the interconnect is digitized in the editor and the required path width is set as a parameter. The digitized coordinates, which in the example in Fig. 3.6e are the coordinates C_1 to C_4, are stored in the data structure along with the path width w. The data volume can thus be approximately halved compared to standard polygons. Paths stored in this way are also easier to modify.

In addition to the path shape in Fig. 3.6e, there are other non-standard shapes whose appearances can be manipulated to meet unorthodox technology constraints. A typical example is the expansion of the thickness with diagonal path segments (some tool providers call these paths segments "padded paths"). This thickness expansion enables the corners of the diagonal path produced by the polygon to lie on the grid. This is a means of preventing rounding errors from occurring when the mask features are produced. The beginning and end of paths with non-standard shapes can be automatically extended. Given that these non-standard path features are also tool-dependent, we shall not dwell further on them here; instead, we recommend the reader to refer to the relevant tool manual.

Edges. It is important to note that modern design tools manipulate shapes as well as parts of shapes. Take, for example, the individual edge segments (C_i, C_{i+1}): these are addressable data items for these tools. This means that individual edges and path

segments can be selected in a layout editor, as well. Useful graphics operations can be performed with these options during layout processing, as we shall explain in Sect. 3.2.3.

Hierarchical Organization of Layout Data

As indicated earlier, layout data are organized in a hierarchy, and this hierarchical organization mirrors the corresponding structural description. Each function block in the structural description—and thus each schematic—is a self-contained subset of the complete layout, which is also called a *layout block*.

The hierarchical layout structure is illustrated as a tree in Fig. 3.7. A layout block (B) can contain components and other layout blocks. The components are described generally in the layout as *cells* (C). Basic components are also called *devices*, whose internal circuitry is typically designed in the *front-end-of-line* (FEOL) in the technology (FEOL is discussed in Chap. 1, Sect. 1.1.3, and demonstrated in Chap. 2, Sect. 2.9.3). The cell shapes (c) therefore are assigned to the layers in the FEOL in the case of devices.

Furthermore, a layout block contains the shapes that form the interconnect layouts produced in the *back-end-of-line* (BEOL) during fabrication (BEOL is introduced in Chap. 1, Sect. 1.1.3, and described in detail in Chap. 2, Sect. 2.8). These shapes are labeled "net shapes" (n) in Fig. 3.7. In contrast to cells and blocks, whose structures are to be found at lower levels in the tree, net shapes are always part of a block. They are said to be "flat" data within a block.

This tree structure of an entire layout exists in both the layout engineer's conception of the design as well as the organization of the design data in the design environment. Here, every layout block (and often every circuit schematic) is typically stored in a separate directory containing specific file formats. The exact data organization depends on the tool used.

While the shapes in the relevant layers are the main focus of interest when generating masks, the layout designer normally works with this layout tree structure,

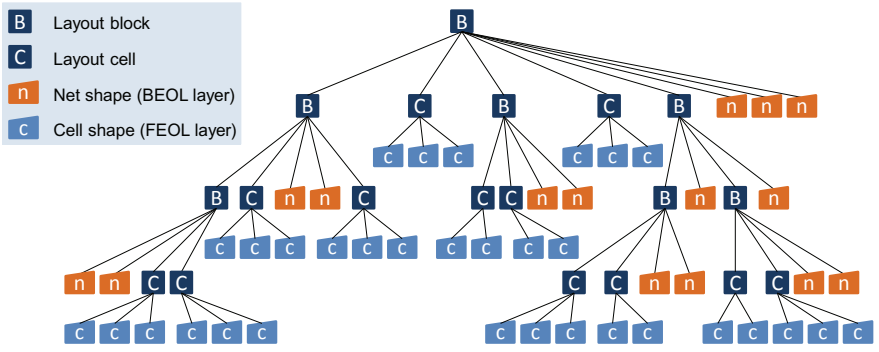

Fig. 3.7 Layout data structure derived from the hierarchical structural description. Layout blocks (B), which are a hierarchical subset of the complete layout, contain other blocks (B), cells (C) and net shapes (n)

and for good reasons. The physical design is considerably simplified by working at higher levels in the hierarchy, as it means the layout designer does not need to handle individual device shapes and parts of subblocks. What's more, thinking in functional units is supported by the hierarchy and the layout designer always has a clear insight into the circuit topology. This "global view" helps further the goal of an optimized final layout configuration.

Despite this facility, a layout designer must have a good grasp of the individual layers and the implications of combining them. He/she may also need to work on the polygonal level in certain cases (sometimes referred to as "polygon pushing"), or at least must take a closer look at it. We demonstrate this in the next section with some practice examples.

3.2.2 *How to Read a Layout View*

A small excerpt from a typical layout is shown in the top part of Fig. 3.8. We will next go through this example step by step, to learn how to "read" a layout.

The layout detail shown in the top of Fig. 3.8 is based on the CMOS standard process we discussed in Chap. 2 (Sect. 2.9). The bottom part of the diagram contains a sectional view of structures generated from this layout. (We use the same colors here as in Chap. 2, Sect. 2.9). If the layout view (Fig. 3.8, top) is a vertical view, the sectional view (Fig. 3.8, bottom) can be thought of as a "horizontal" view of the circuit, along the cutting line as identified in the layout (Fig. 3.8, top). Utilizing the

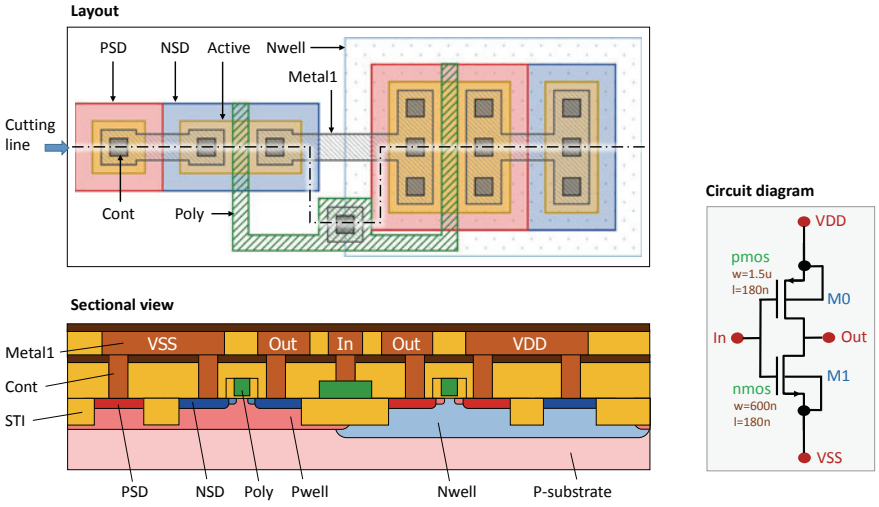

Fig. 3.8 The layout of a simple CMOS inverter (top), as shown in a typical layout editor, and the corresponding sectional view (bottom) and schematic (right)

circuit diagram (see Fig. 3.8, right), we see that the layout is a simple configuration comprising an NMOS transistor and a PMOS transistor. The sectional view shows that the connection points for these transistors on the silicon surface are in part interconnected and contacted with the first (bottom) metal layer (Metal1). This is, in effect, a "circuit" at this stage.

Using today's layout design tools, the engineer is only presented with a layout view (Fig. 3.8, top). As the tool does not generate or present the sectional view, he/she only sees the (two-dimensional) layout structure. Consequently, although the engineer has to "work" in two dimensions, it is useful—sometimes unavoidable—to "think" in three dimensions. "Reading" a layout means recognizing the devices and their electrical connections on the chip and imagining how they are shaped physically. Although this may initially seem a daunting task, there are techniques that can be learned that make it easier, as we discuss next.

As a first step, the devices need to be identified: this is done by examining the FEOL layers. We start by focusing on the drawn layer representing the "active" areas, which define sections of the chip surface without field oxide (shown ocher in Fig. 3.8, top). Normally, a drawn layer is assigned for these regions, but designations differ greatly from manufacturer to manufacturer. The drawn layer in question is called "Active" in our layout example in Fig. 3.8 (top). The mask "STI" (shallow trench isolation) is produced from this layer by negation, i.e., the shapes in "Active" define the regions that remain unaffected by STI.

In addition to these active surfaces, we are looking for polysilicon ("Poly", shaded green in Fig. 3.8, top) as both layers combined indicate a transistor. Specifically, wherever shapes from these two layers cross, there is a channel of a field-effect transistor (FET). It is often possible to identify most instances in a layout this way, given that FETs are by far the most common basic devices. The aforesaid applies to digital circuits and to most analog circuits.

Figure 3.8 shows that "Active" (ocher) and "Poly" (shaded green) cross in two places. We have therefore two FETs in our example.

Figure 3.9 depicts these two transistors separately (these transistors were introduced in Chap. 2, Sect. 2.9.3, see also Fig. 2.35e) by illustrating the layout and sectional views with labeled source (S), drain (D) and bulk (B) contacts. Bulk (aka *backgate*) contacts "belong" to the transistor layout as they define the potential of the well or the substrate, respectively.

The difference between NMOS-FETs and PMOS-FETs is that the bulk areas of PMOS-FETs are defined in processes with a p-substrate by a drawn "Nwell" layer (spotted pale blue, see Fig. 3.9, top right). The transistors outside Nwell areas are therefore NMOS-FETs. Bulks for NMOS-FETs are either the p-doped substrate of the wafer (for single-well processes) or the areas with a fabricated Pwell layer (twin-well processes). Either of these two scenarios could occur in our layout example, as the Pwell-doped areas could be derived from the drawn "Nwell" layer by negation (as we have seen in Chap. 2, Sect. 2.9.3) and would not then appear as a separate (drawn) layer in the layout.

Now that we recognize NMOS-FETs and PMOS-FETs, the dopant types (n or p) of the blue and red layers forming the source and drain areas should become clear

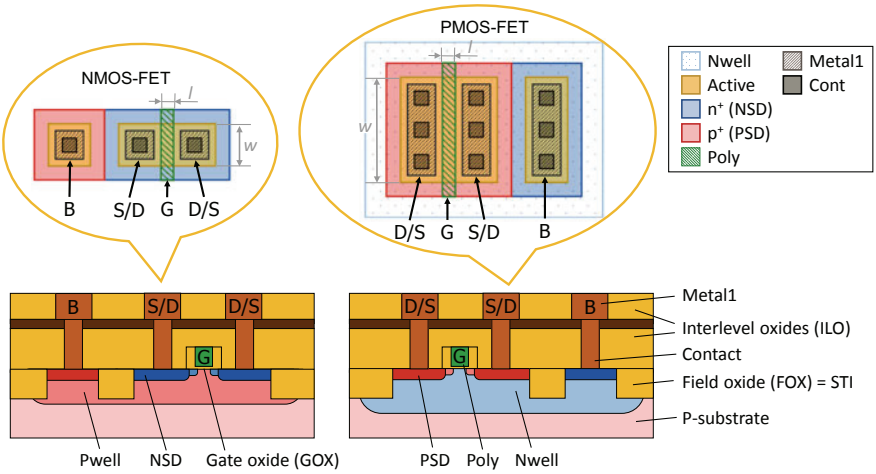

Fig. 3.9 The layout and the corresponding sectional views of the two transistors in Fig. 3.8 with marked contacts (D/S: drain/source, B: bulk) and gates (G). Transistors can be identified in any layout structure by focusing on crossings of the active areas (layer "Active", here depicted in ocher) and the polysilicon (layer "Poly", here shaded green)

as well. Additionally, n- and p-doping are reflected in the layer names: These n^+ and p^+ implanted layers are respectively labeled "NSD" and "PSD" in our example.

Finally, the BEOL layers, which connect the devices, are to be considered. Contacts and vias, for example, which are small, uniform squares in state-of-the-art processes, are generally easy to pick out. There are also metal features, which must always cover the contacts and vias and which form interconnects above the devices. In our layout example in Fig. 3.8, we have contact holes in the drawn "Cont" layer (dark gray) for contacting the source, drain and bulk regions. The layout of the interconnects are shown as the drawn layer "Metal1" (shaded bright gray). The gates, electrically connected by poly, have a common contact in metal, which is contacted to poly by the same "Cont" layer.

The two transistors are connected to form a logic inverter. The circuit schematic for the example is depicted on the right in Fig. 3.8.

3.2.3 Graphics Operations

A wide range of edit commands and graphics operators are available in modern layout editors. We shall only concern ourselves here with operators for manipulating and selecting shapes. Convenience commands for configuring a number of elements,

such as the "Distribute", "Align", and "Compact" commands, are not dealt with here as they are well-known and intuitively understandable.

Interactive Shape Editing
Layout editors feature all of the standard graphics commands that we are familiar with in other drafting software. Shape commands that are generally available include "Add", "Delete", "Move", "Copy Paste", "Flip", and "Rotate". Layout editors also offer different user concepts specific to the layout process that make working with the tools easier. Data entry can be made with the mouse; numerical and text data can also be entered using the keyboard, and so on.

In addition to these standard functions, other commands are available for working with shapes when designing the layout:

- Stretching a shape by shifting a subset of its edges or corners ("Stretch"),
- Changing polygons by cutting out, truncating and attaching rectangles or more complex polygons (e.g., "Notch"),
- Merging overlapping shapes into one shape ("Merge"),
- Splitting polygons along (any) intersecting lines ("Split").

Logical Linking of Layers
Boolean operators from the field of mathematical algebra can also be applied to shapes in different layers. They are very important and powerful operators that "logically link" the "content" of these layers. While they are deployed sometimes in physical layout design, their main use is in the *design rule check* (*DRC*) to identify specific layout constellations for checking (Sect. 3.4 and Chap. 5, Sect. 5.4.5) and in the *layout post process* (Sect. 3.3) to produce mask data. We demonstrate the following, general standard logic operators in Fig. 3.10:

- OR: produces the geometrical union of two layers.
- AND: produces the geometrical intersection of two layers.
- XOR: produces the union minus the intersection of two layers.
- ANDNOT: generates the "geometrical difference" between two layers. Everything that is in the second layer is "punched" out of the content of the first layer.

The upper portion of the figure shows a simple sample layout. This layout consists of four rectangular shapes, two of which belong to a "red" layer and the other two to a "blue" layer. The results of the operations are written in a new layer "x", shown in gray at the bottom of Fig. 3.10.

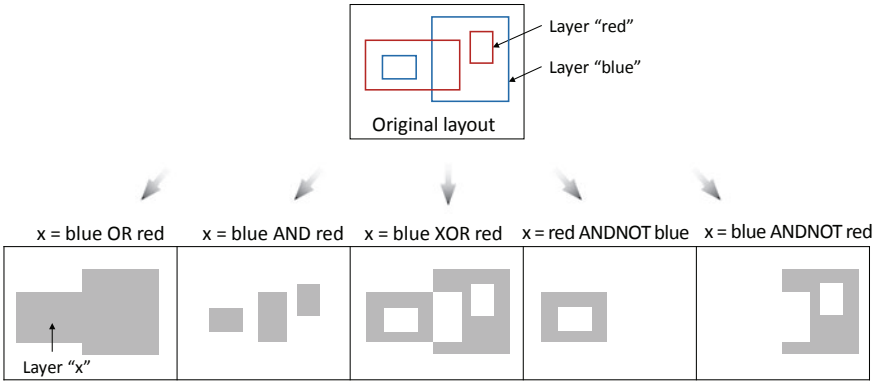

Fig. 3.10 Logical linking operations applied to four shapes on two layers (top) using the standard logic operators OR, AND, XOR, and ANDNOT (bottom, left to right)

Select Operations

Shapes that satisfy a specific criterion in a layer can be picked with select commands. In Fig. 3.11, we demonstrate some key selection criteria based on specific relationships between the shapes in the specified layers:

- INCLUDE: Selects shapes in a layer that overlap in any way with shapes in another layer.
- OUTSIDE: Selects shapes in one layer that do not overlap with shapes in another layer.
- INSIDE: Selects shapes in a layer that are fully covered by shapes in another layer.
- ENCLOSE: Selects shapes in a layer that fully cover shapes in another layer.
- CUT: Selects shapes in a layer that share a portion of their surface area (but not their entire surface area) with shapes in another layer.

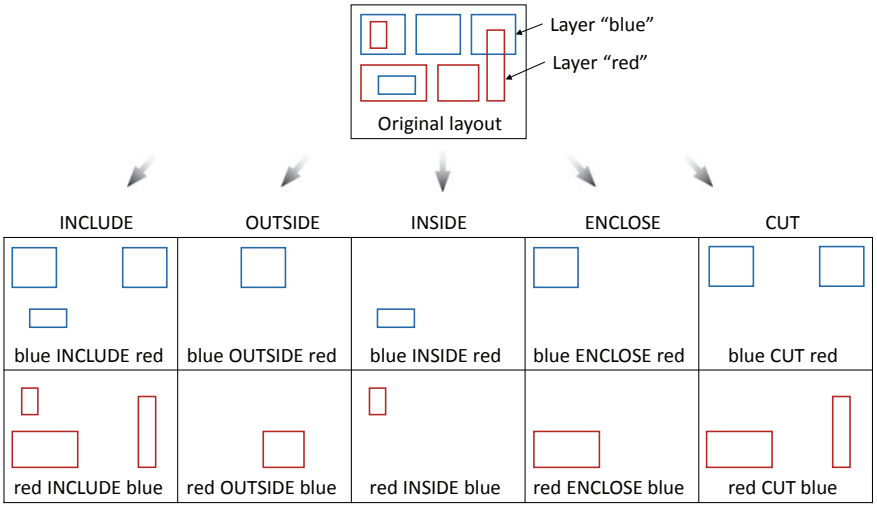

Fig. 3.11 Selection commands for filtering shapes from layers based on geometrical relationships

In contrast to the linking operations, no new geometries are created by the selection commands; instead, existing geometries that meet the criteria are selected (i.e., identified). The results can be saved in a new layer as required. The selection commands are of great interest for the DRC, as layer subsets of interest can be identified this way (Sect. 3.4.3, Example 1).

Sizing Operators

We introduced the *sizing* operator in Chap. 2 (Sect. 2.4.2) when we discussed preemptive edge shifts. (The structure's boundary lines are shifted outwards by a specific value or inwards in order to compensate for shrinking/enlarging effects that can occur in subsequent structuring process steps.) Sizing is also very useful in the DRC to check layouts for compliance with more complex design rules (Sect. 3.4.3, Example 2). Furthermore, sizing can be applied to "clean up" layouts, as we shall explain next.

A polygon is modified with the sizing operator by shifting all edges perpendicular to the edge alignment by a specific value. The polygon is enlarged if the edges are shifted by positive values; this operation is called *oversizing*. Whereas the polygon shrinks if the values are negative; this latter operation is called *undersizing*.

Sizing has several noteworthy properties that must be understood, as they can produce unfortunate and unexpected results if you are not aware of how they work. Having said that, you can also leverage these properties to produce specific effects that are helpful. We will take a closer look at these effects now.

Uneven growth. Oversizing by a value s causes the corners of a polygon to be shifted by a distance $v \cdot s$, where $v > 1$, i.e., the *corners* are always shifted from their original positions by *more* than the shift value s, e.g., $v = \sqrt{2}$ for right angles. For acute angles (angles <90°), the value v is greater than $\sqrt{2}$ and can theoretically be extremely large.[1] This effect is a generic defect in sizing, as "even" growth in all directions is the desired outcome in most cases (Fig. 3.12, left).

Ideally, we would like circular arcs at the corners (a circle defines a set of points with identical distances to the corner). But as we know, arcs cannot be modeled in the data structure. Oversizing can however be configured in some tools such that the corners can be "beveled" with additional edges to approximate a circular arc as per Fig. 3.6c. Two examples are shown in Fig. 3.12 (right).

Rounding error. Another difficulty with the sizing operator is that, in the case of inclined edges (see Fig. 3.12, bottom), the corners are not placed on the grid. This causes rounding errors because integer values are used for the coordinates. While these rounding errors can often be ignored, the angles w.r.t. the coordinate system may be altered (Fig. 3.13c), producing unwanted results in some scenarios. For example, design rules may be violated by this effect, which would not have occurred with mathematically correct sizing.

[1] Acute angles are often not allowed in layouts because the edges in question are treated as being "opposite" by DRC tools and are flagged as width rule violations. Even if this is not a problem for fabrication, these cases should be avoided to minimize the work involved in evaluating a DRC.

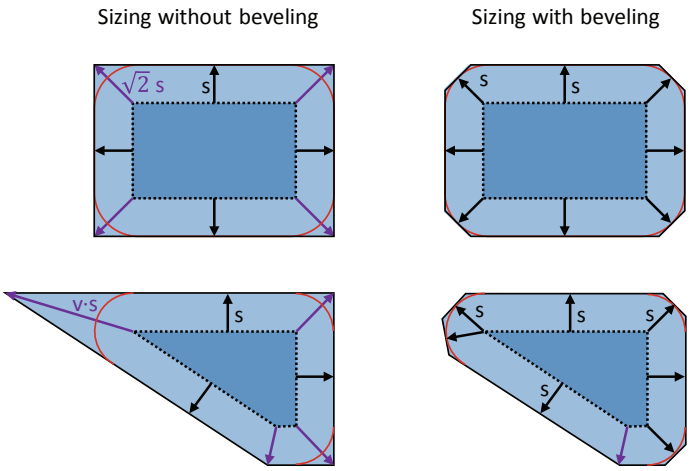

Fig. 3.12 Oversizing without (left) and with beveling corners (right)

Irreversibility. If two sizing operations are performed immediately one after another, with the same value but in opposite directions, the final result may not be the same as the original structure. There are a number of reasons for this, as illustrated in Fig. 3.13:

- Small polygons disappear during undersizing (also narrow ribs), Fig. 3.13a,
- Small holes in polygons disappear during oversizing, Fig. 3.13b,
- Rounding errors cause shape changes, Fig. 3.13c.

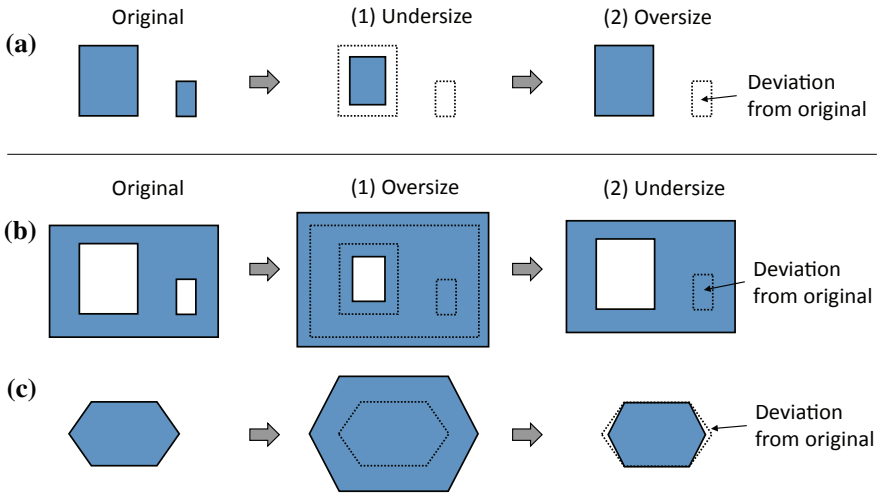

Fig. 3.13 The effects of two sizing operations with the same absolute value but different signs, i.e., sizing in opposite directions, that can produce polygons which deviate from the original shape

Cleaning up layouts. If sequences of sizing operations and logical links are used to create certain layout structures in the layout post process or in the DRC, these steps may produce unwanted shapes caused by rounding errors. These unwanted shapes are often very small. Hence, the depicted effects in Fig. 3.13a, b could be used to positive effect to eliminate such small artifacts.