

Ann Lanfri

PHLED45: An Enhanced Version of Caesar Supporting 45⁰ Geometries

APPROVED:

Paul W. Sandberg

Assistant Professor in Charge of Major

Curtis R. Cook

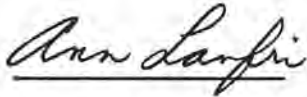
Chairman of Department of Computer Science

Date research paper is completed April 1985

The attached paper describes a project I completed on a graphic editor Phled45. The work was done towards a Master's of Science degree for the Computer Science Department at Oregon State University, Corvallis, Oregon. A version of this paper was presented at the 1984 ACM/IEEE Design Automation Conference. Reprints of the paper, entitled "Phled45: An Enhanced Version of Caesar Supporting 45° Geometries", are available from the IEEE Computer Society.

I have added two sections to my original paper on Phled45. In the first part I have evaluated how well my original design decisions have been accepted by Phled45's users. (Phled45 is a commercial product having several hundred users.) I have also described new features that I added after my original work for 45° geometries.

The second new section is the UNIX manual pages for Phled45. More details about Phled45, including a tutorial, are available from Metheus-Computervision in the *Physical Design Tools* volume which is part of their *VLSI Tools Manual*.

A handwritten signature in cursive script, reading "Ann Lanfri". The signature is written in dark ink and is underlined.

Ann Lanfri

April 1985

PHLED45: AN ENHANCED VERSION OF CAESAR
SUPPORTING 45° GEOMETRIES

Ann R. Lanfri

Oregon State University
Corvallis, Oregon

Metheus Corporation
Hillsboro, Oregon

ABSTRACT

Caesar is an interactive graphic editor that is used to generate layouts for VLSI circuits. It runs on a VAX-11/780 under the UNIX¹ operating system with Berkeley extensions. Caesar has a unique and simple user interface. Phled is an enhanced version of Caesar that runs on a M68000-based engineering work station. Phled45 is based on Phled; its distinctive feature is the ability to enter layout containing 45° angles. The design considerations for implementing the capability to edit 45° shapes are discussed. The details of the implementation are presented.

¹UNIX is a trademark of Bell Laboratories.

INTRODUCTION

History

Caesar. Caesar is a color graphic editor that is used to edit Manhattan (edges are vertical or horizontal) layouts for VLSI circuits. It was written at the University of California at Berkeley by John Ousterhout [1] in the C language and contains about 15,000 lines of source code, including comments. Caesar runs under the UNIX operating system with Berkeley extensions on a VAX-11/780. It uses a bit pad with puck, a color display, and a video terminal.

In Caesar, as in most other layout editors, a design is built hierarchically from cells. Each cell may contain other cells (subcells) and also the geometries that define a mask pattern. There are commands in Caesar for editing cells in context; there are also commands for placing, rotating, deleting and arraying cells.[2]

There are several features in Caesar that make it unique as compared to other layout editors.[3] For the purposes of this paper, the most important of these features is the way that mask geometries are entered.

Most layout editors are object-based. The objects that are represented internally can include polygons, wires and stretchable components. The commands that a user issues to edit these objects may be different for different types of objects. The user must therefore be aware of the type of the object that he is editing even though two different objects may result in the same mask pattern. With this kind of editor, the user manages the internal data structures by his editing actions.

In Caesar, the shapes that define a mask pattern are entered by the "painting" operations. A rectangular box is positioned in the design area using the puck and bit pad; a crosshair cursor selects the mask layers to be edited. The area that is inside the box can be filled with the mask layers ("painted"), erased or copied ("yanked"). Unlike object-based editors, Caesar itself is responsible for managing the internal representation of the geometries. Therefore, the user has a simple way to create the mask pattern. He need not be concerned with how the geometries were entered nor by how they are represented internally in Caesar.

Caesar stores the mask pattern ("paint") internally as rectangles. For any given shape, there is a unique internal representation, regardless of how the shape was entered. Thus, no matter how many times the data base is edited, its representation will not fracture into more and more pieces. The unique representation is achieved by not allowing overlapping paint and by merging the data base in a specific way: each rectangle is as wide as possible (maximal horizontal strip) and any two rectangles that share an entire horizontal edge are merged into one rectangle.

Phled. Phled (Physical layout editor) is an enhanced version of Caesar that has been transported to a M68000-based engineering work station (Metheus $\lambda 700$ series). The graphics part of the work station consists of a high-resolution (1024 by 768) color monitor that is driven by a bit-sliced display controller. The pointing device is an optical mouse.

Like Caesar, Phled is written in the C programming language. The source code consists of about 20,000 lines, including comments. Phled has many features that are not present in Caesar.[4] Here, it is the similarities between Caesar and Phled that are the most important: Phled uses the same user interface as Caesar (the box), the same internal representations, and the same data base management techniques.

Phled45. Phled45 is a version of Phled that supports the entry of both Manhattan and 45° shapes. It consists of approximately 23,000 lines of C code, including comments. This paper will describe the process of designing Phled45 from Phled. First, I will discuss the motivation for extending Phled to have the capability for entering 45° designs.

Manhattan vs. Non-Manhattan Layouts. There was a deliberate decision made in the design of Caesar to support only Manhattan geometries. Restricting the data base to rectangles can greatly simplify programs that deal with layouts. As I will discuss later, two operations that Caesar performs as part of its data base management are determining whether two shapes intersect and clipping one shape against another. Both of these operations are simple for rectangles but are more difficult (require more complex code and thus execution time) for general polygons.

There are penalties incurred by a Manhattan-only design technique. First, the layout may consume more area. This can be a significant factor for some designs.

A designer has two concerns about chip area. First, he wants to make the chip as small as possible because the smaller the chip area is, the more chips can be put on one wafer. The second concern is to put as much functionality as possible in a certain size chip. Since manufacturing costs are fixed for a certain die size and process, putting more gates on the same size chip decreases the cost per gate. Many designers use 45° shapes to save space and to squeeze in as many gates as possible.

Another disadvantage of Manhattan design is that it may increase the length that signals must travel on the chip. The use of 45° angles may allow the designer to shorten this distance.

Arbitrary angles are also used by some designers in their layouts. With such angles the user is able to express features such as circles more closely and to minimize the length of runs. But most commercial designers have restricted their layouts to Manhattan and 45° geometries even if their layout editors support all angles.[5]

DESIGN GOALS FOR PHLED45

The goals in designing Phled45 from Phled were to implement the ability to edit 45° shapes while minimizing the impact on three areas: the user interface, the editor's performance, and the source code itself.

It was important that a Manhattan user of Phled45 be able to enter data in the same way as is done in Caesar. In addition, it was also important that the method of entering non-Manhattan shapes be consistent with the editor's user interface.

There were similar performance goals. The first was that a Manhattan user not be penalized significantly in performance by the additional capability. In addition, entering non-Manhattan shapes had to be done in a reasonable amount of time.

Finally, the changes had to be done without massive rewriting of the source code. If radically different algorithms were used, the task of adding 45° capability could have been more work than designing a new editor.

IMPLEMENTATION DETAILS

The implementation of Phled45 was divided into three parts. First, we chose the user interface for entering 45° shapes. Next, some existing data structures were modified and new ones created to support the new capability. And finally, the algorithms for the internal management of rectangles were modified and extended to handle 45° shapes.

User Interface

We ruled out radical changes in the user interface for Phled45 to keep it as close as possible to that of Caesar. For example, entering 45° shapes by digitizing the vertices of a polygon was not a reasonable option. The idea of using a box to paint and erase was retained.

We chose to use a rotated box for entering 45° geometries. (See Figure 1.) The box can be rotated 45° to the left or to the right about its center. The vertices of the box are on integral coordinates and the width and height of the rotated box are irrational.

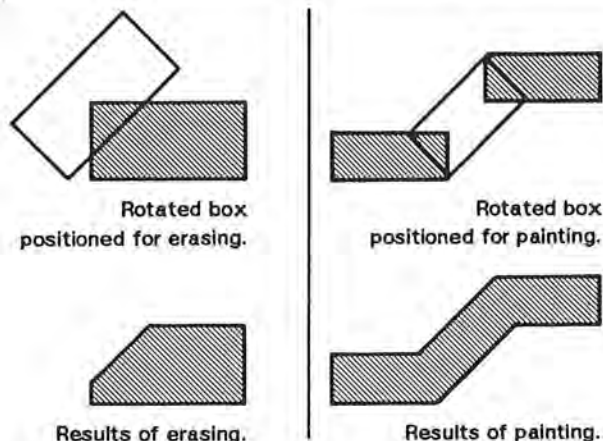


Figure 1. Painting with Rotated Box

To enter a 45° shape, the Manhattan box is rotated 45° by pushing a mouse button. Once the box has been rotated, it is manipulated like the Manhattan box: the rotated box can be positioned and its size modified with the mouse buttons and the user can paint, yank, or erase the area inside the box. The box can be rotated 45° back to a Manhattan box by again clicking a mouse button. The user can thus enter 45° shapes without typing commands from the keyboard. This is consistent with the way Manhattan shapes are entered in Caesar.

Data Structures

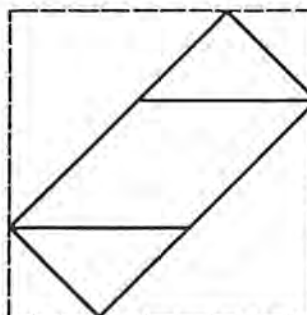
The existing data structure for geometries (a rectangle) was modified and new data types created to handle the 45° capability for the box, painting operations, and drawing. However, no changes were made in Phled45 to handle the non-paint structures like cells and labels. Labels are still rectangular and although a cell may contain 45° shapes, its bounding rectangle (i.e. the smallest rectangle entirely enclosing the cell's shapes) is still rectangular. Cells may be rotated only in increments of 90° as in Caesar, so there were no changes necessary to their data structures.

Rotatable Box. The box has several different functions in Caesar. The designer uses it to position cell instances, add labels, and define areas to be painted or erased. Internally, the box also determines what structures from what cells are possibly affected by a change to the data base. This last use is important in Caesar; it allows the editor to quickly reject entire cells and pieces of paint from consideration.

In Phled45, the bounding rectangle of the box (this is the box itself when it has not been rotated) positions labels and cell instances. Internally, the bounding rectangle also screens cells and "paint" for the editor's consideration. Only the painting operations use the actual shape of the rotated box.

So, the first part of the box's data structure is a rectangle representing its bounding rectangle. The next part is a flag indicating whether the box has been rotated and a list of the rotated box's vertices.

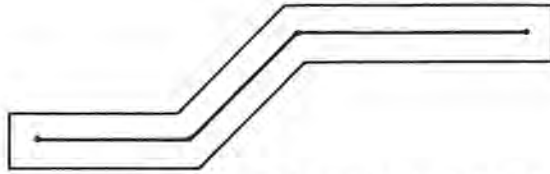
As will be seen in the next section, paint in Phled45 is represented by a trapezoid. For painting operations where the exact shape of the box is required, the box is fractured into a list of trapezoids that can be input to the various data base manipulations. (See Figure 2.) For an unrotated box, this list has one shape that is the bounding rectangle itself.



The rotated box is fractured into 3 trapezoids only when necessary for painting operations. Otherwise its bounding rectangle is used.

Figure 2. Rotated Box

There is an alternative to using the box for entering paint. Phled45 supports a "wire" for entering a constant width path. This is not a wire in the true sense since a wire is stored by its path and width. The user defines the center line of the path and the appropriate paint is automatically generated. (See Figure 3.)



An alternative way of entering paint is to define a "wire".

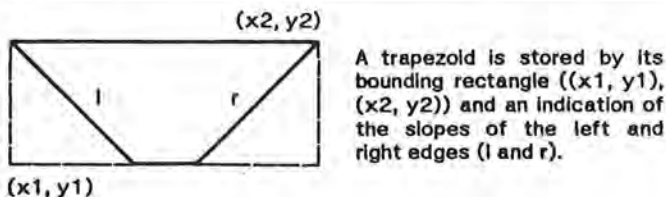
Figure 3. "Wires"

Paint. As I mentioned previously, when a rectangle ("paint") is added to the data base in Caesar, maximal horizontal strips are created and then merged wherever possible. To minimize the impact on the data base manipulation algorithms, the internal representation for 45° shapes needed to be something as close as possible to rectangles. This ruled out the possibility of using general polygons.

We chose a special type of trapezoid: a trapezoid whose top and bottom edges are horizontal. The length of the top or bottom edge can be 0, i.e. the trapezoid can be a triangle. Because the top and bottom edges are horizontal, this trapezoid fits very well into the existing data base algorithms. The sides of the trapezoid can be vertical, tilted 45° to the left, or tilted 45° to the right.

The idea of using this special type of trapezoid is not new. The horizontal top and bottom edges are well suited for scan line conversions of geometries, and these scan lines are used by certain design rule checkers and circuit extractors. A trapezoid of this type is also used by some E-beam IC mask-making machines. Jackson [6], gives reasons for using this data type as a primitive for graphics applications. In Newell and Sequin [7], an algorithm for converting a polygon into a set of non-overlapping trapezoids is presented.

A trapezoid is described by its bounding rectangle and an indication of which way the left and right edges tilt. (See Figure 4.) Rectangles are also represented by trapezoids: a rectangle has vertical left and right edges and its bounding rectangle is the rectangle itself. The editor can tell a rectangle from a trapezoid if it is necessary by looking at the slopes of the sides.



A trapezoid is stored by its bounding rectangle ((x1, y1), (x2, y2)) and an indication of the slopes of the left and right edges (l and r).

Figure 4. Trapezoid Data Structure

A trapezoid is treated as a rectangle (only its bounding rectangle is used) whenever possible. For example, the bounding rectangle of a cell is calculated from the bounding rectangles of the objects inside it, not from the objects themselves. The actual vertices of the trapezoid can be derived whenever it is

necessary by adding (or subtracting) the height of the trapezoid to the bounding rectangle's x coordinates.

Drawing. Caesar was designed to be easily transported to systems having any of several kinds of color displays. Since not all of these displays support their own clipping, Caesar does its own clipping. Each rectangle drawn in Caesar is clipped to the area affected by the command.

Phled45 takes advantage of a particular set of hardware that has a display controller available for clipping. Phled45 sets a clip window before sending a series of shapes to be drawn and does not need to do its own clipping. All shapes whose bounding rectangle intersects the the area to be redrawn are sent for drawing.

There are no specialized routines available for drawing trapezoids in the display controller, so the general polygon drawing routines are used. Each trapezoid is converted to a polygon data structure. Then the polygon is transformed into a screen coordinate system for drawing.

Because drawing rectangles is faster than drawing polygons on the hardware used by Phled45, trapezoids that are actually rectangles are singled out and drawn as rectangles.

Paint Data Base Management

Caesar adds paint to the data base in three steps.[3] First, the paint is clipped against any existing paint; this eliminates the possibility of overlapping paint. Next, the clipped paint is added to the data base: maximal horizontal strips are created by splitting and horizontally merging the rectangles. The final step merges the rectangles vertically.

Erasing in Caesar is similar to painting. First, the existing rectangles that intersect the area to be erased are clipped against that area. Then the rectangles formed by the clipping are added to the data base and the data base is merged vertically in the same way as is done for painting.

Manipulation of paint in Phled45 follows the same set of steps. But handling 45° shapes is more complex. The following paragraphs describe how intersection, clipping and merging are done in Phled45 and how this differs from Caesar.

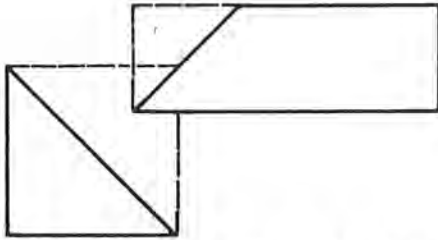
Intersection. In Caesar, checking for intersection between two shapes is simple: the shapes are always rectangles. In Phled45, the shapes to be checked are trapezoids. It is not enough to check their bounding rectangles for intersection: the bounding rectangles may intersect, but the shapes may not. (See Figure 5.)

The work involved in finding the intersection of two trapezoids is similar to the calculations required for clipping. So instead of duplicating effort, trapezoids whose bounding rectangles intersect are sent to the clipping routines (see "Clipping" below). Actual intersection is determined as the clipping proceeds.

Clipping. Caesar clips in two phases, horizontal first, then vertical. The horizontal clipping is done first to ensure maximal horizontal strips.

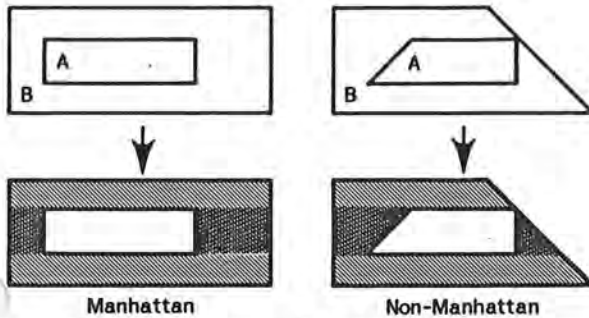
For many cases in Phled45, clipping trapezoids is about as simple as clipping rectangles. (See Figure 6.)

Horizontal clipping of trapezoids is quite similar to that of rectangles because both shapes have horizontal edges.



The bounding rectangles intersect but the shapes do not.

Figure 5. False Intersection



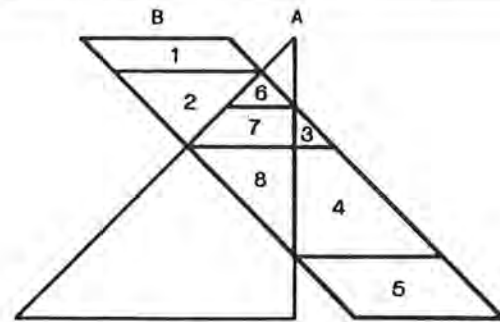
Clipping B against A creates 4 shapes: the top and bottom are from horizontal clipping; the left and right are from vertical clipping.

Figure 6. Clipping

Vertical clipping of trapezoids can be more complicated than clipping rectangles. For example, consider Figure 7. The vertical clipping proceeds one edge at a time. First, Trapezoid B is clipped against the left edge of Trapezoid A. Trapezoids 1 and 2 are split off from B since they are definitely outside of A. The original Trapezoid B has been fractured by this splitting process into those trapezoids that are to the right of Trapezoid A's left edge. Each of these remaining parts is clipped against the right edge of Trapezoid A. This causes Trapezoids 3, 4, and 5 to split off since they are outside of A. The remaining Trapezoids 6, 7 and 8 represent the area inside of Trapezoids A and B.

Depending on the type of data base operations being performed, the inside or outside or both lists of trapezoids will be used. For example, if Trapezoid A is in the data base and Trapezoid B were to be added, the outside trapezoids (1, 2, 3, 4, 5) would be the paint from B actually added.

If, however, B were a trapezoid defining a region to be copied, the inside trapezoids (6, 7, and 8) would be paint copied.

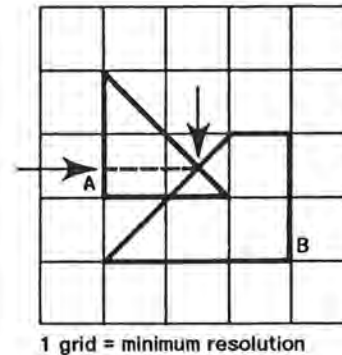


Clipping B against A can result in several trapezoids.

Figure 7. Vertical Clipping

As mentioned in the previous section about intersection, Phled45 attempts to clip shapes whose bounding rectangles intersect. As the clipping proceeds, the actual intersection is computed. The clipping operation will return an empty list of shapes if the shapes did not actually intersect.

There is a new problem introduced by the clipping of 45° shapes. The coordinate system of Phled45, like Caesar's, consists of whole numbers. In the process of clipping, it is possible to create a shape whose vertices do not lie in the editor's coordinate system. (See Figure 8.)



1 grid = minimum resolution

When A is erased with B, the clipping of A could fracture it into shapes whose vertices (see arrows) cannot be represented in Phled45's coordinate system.

Figure 8. Illegal Trapezoids

This can be further illustrated by looking at an equation for the intersection of two non-parallel 45° lines in terms of a point (x_1, y_1) from the first line and a point (x_2, y_2) from the other line:

$$\left(\frac{y_2 + x_2 - y_1 + x_1}{2} \right), \left(\frac{y_2 + x_2 + y_1 - x_1}{2} \right)$$

If the numerator is odd, the resulting point will be fractional and not in the coordinate system. The numerator will be even if the sum (or difference) of the x and y coordinates of the points is forced to be even. So Phled45 restricts the possible places the box can be positioned to only those points whose sum (or difference) of x and y coordinates is even. Because

Phled45 maintains more precision in the data base than is allowed to the user, he is not aware of this restriction.

Merging. Caesar merges two rectangles horizontally if their non-horizontal edges touch, i.e. if the left x coordinate of one is the same as the right x coordinate of the other. (See Figure 9.) The same is true in Phled45 except that it is more difficult to tell if two 45° edges touch. If the bounding rectangles touch and the edges are Manhattan, the shapes do touch. Otherwise, the editor checks the right edge of one shape to see if it is tilted in the same direction as the left edge of the other shape. If so, the next check determines if the two edges are collinear.

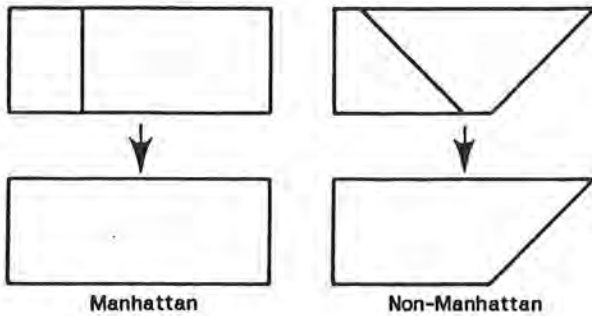


Figure 9. Horizontal Merging

Caesar vertically merges two shapes that touch if both the left and right edges have the same x coordinates. (See Figure 10.) In Phled45 for non-Manhattan edges, the editor must calculate the actual x coordinates using the bounding rectangle of the shape and its edge types.

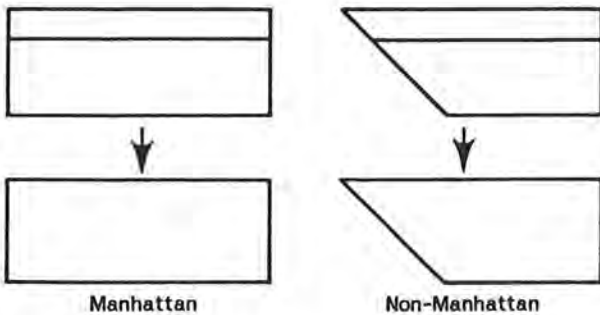
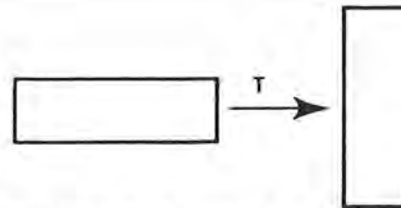


Figure 10. Vertical Merging

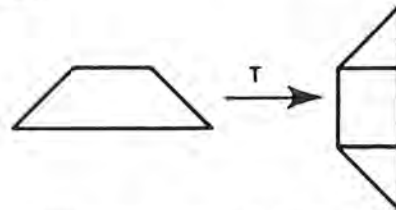
File I/O. Trapezoids and rectangles (trapezoids with two vertical edges) are read in and stored on the same lists in Phled45. They are separated from each other when the file is written. An entry in the file indicates whether it contains 45° shapes; any tools that use the design can tell if they can take advantage of the faster operations often possible for Manhattan-only designs.

Transformations. In Caesar, a subcell has a transformation matrix that maps its coordinates into the coordinate system of the root cell. Similarly, there is an inverse transformation that takes the root cell coordinates into the coordinate system of the subcell. This mechanism makes it easy to allow editing in context.

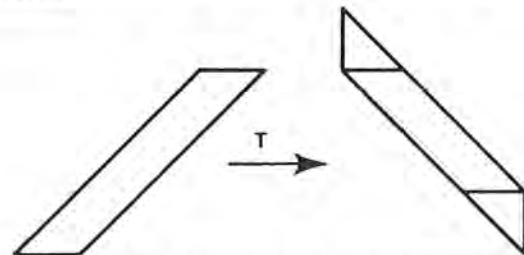
Transforming a rectangle in Caesar is done by applying a transformation matrix to the rectangle's coordinates. This yields a rectangle in the other coordinate system. In Phled45, the same matrices are used but this process is more complex because trapezoids are being transformed. For example, see Figure 11: if the trapezoid is to be rotated by an odd multiple of 90° the resulting shape would not be a legal trapezoid because its top and bottom edges would no longer be horizontal.



No combination of mirroring and rotation in 90 degree increments can transform a rectangle into any shape but a rectangle.



Some trapezoids can be transformed into a rectangle and 2 trapezoids.



Some trapezoids can be transformed into 3 trapezoids.

Figure 11. Transformations

In Phled45, the trapezoid to be transformed is first fractured, if necessary, as illustrated in Figure 11. The bounding rectangle for each resulting shape is then transformed by the matrix. Each transformed shape is turned into the correct shape by looking up its type in a table that is based on the original shape and the particular transformation applied. There are eight possible transformations and four combinations of non-vertical left and right edges, so this table has 32 entries.

Trapezoids that are actually rectangles are handled as a special case and transformed as in Caesar. They are never fractured.

EVALUATION OF IMPLEMENTATION

The next section discusses how well the implementation of Phled45 met our original design goals.

User Interface

From Phled45's point of view, the easiest shape for entering 45° paint would have been a trapezoid. Since the data base contains trapezoids, all of the routines necessary for handling a trapezoid were already present. Some functions for the user to change the size and shape of a trapezoidal box would have been needed, but they would have been straight-forward to do.

The resulting editor would not have been as easy to use, however. The user would have had to think about his layouts in terms of trapezoids. This would not have been desirable since an important feature of Caesar is that the user does not have to know anything about the internal representation of the paint.

The editor's users have readily accepted entering 45° shapes by manipulating a rotated box. The rotated box preserves the simple, easily learned user interface of Caesar. Users who want to define constant width shapes containing 45° angles can do so with a pseudo-wiring command.

No changes have been made in the Manhattan user interface. A Manhattan user of Phled45 notices no differences in entering layouts.

So the goal of entering 45° shapes in a manner consistent with Caesar has been met with Phled45. In addition, the goal of not changing the operations for a Manhattan user has been satisfied.

Manhattan/Non-Manhattan Performance

Because Caesar runs on such different hardware, it would not be meaningful to compare Caesar's performance to that of Phled and Phled45. Table 1 shows the comparison between the amount of execution time required for some selected operations in Phled and Phled45. The times are normalized to the operation in Phled. For example, it takes three percent longer to read a rectangle in Phled45 than in Phled. Reading a trapezoid takes 26 percent more execution time than reading a rectangle in Phled.

The "rectangle" column in Table 1 shows the best case figures (all shapes are rectangles). The last column contains the worst case figures (all shapes are trapezoids). In practice, a design might contain between 20 and 30 percent trapezoids. The performance penalties for supporting 45° shapes are incurred only in those areas that actually contain the shapes. So, the actual performance of Phled45 for a typical 45° layout would be closer to the Manhattan performance.

It takes slightly longer for all Manhattan operations in Phled45. This penalty is never more than about ten

Table 1.

PHLED45 PERFORMANCE (Normalized to operation in Phled)		
	Rectangle	Trapezoid
Drawing	1.01	2.73
Reading	1.03	1.26
Writing	1.01	1.25
Moving box	1.13	1.38*
Painting	1.11	1.55
Erasing	1.07	1.15
Yanking	1.09	1.13

* rotated box

percent. All of the Manhattan operations are the same in Phled45 as they are in Phled. The extra time is required to determine if the shapes are actually Manhattan.

Next, I will discuss the performance of Phled45 for non-Manhattan operations as illustrated by the third column in Table 1.

Box Manipulation. The editor takes approximately 40 percent longer to move the rotated box than the Manhattan box. In general, all of the box manipulation operations are more difficult for the rotated box. It is harder to calculate widths and heights for the rotated box and to compute how to rotate the box. The rotated box consists of a bounding rectangle and a set of polygon vertices and both must be updated when a new box is computed.

Most of the time, the user is manipulating the box interactively. Modifying the rotated box is still a small part of the user's time while he is editing layouts. The extra time to manipulate the rotated box is not usually significant.

File I/O. About 25 percent more time is required to read and write a trapezoid. This is mainly due to two factors. First, as a trapezoid is read in, it is checked to be sure that it describes a legal shape. Second, there are two more fields to read and write (the slopes of the trapezoid's sides) than for a rectangle.

Paint Operations. Adding a trapezoid to the data base takes about 50 percent longer than adding a rectangle. When the rotated box is used to enter "paint", it is fractured into three trapezoids. The time to add these trapezoids is almost three times as long as adding one rectangle to the data base in Phled.

Straightforward cases of erasing or yanking a trapezoid take about 15 percent longer than the same operation for rectangles in Phled. It has already been seen how complicated clipping can become for the paint operations on trapezoids. Situations like the one shown in Figure 8 would take longer.

Drawing. The time to display a trapezoid in Phled45 is about three times longer. This is partly due to the fact that it takes longer for the display controller to draw a polygon than a rectangle. The other factor that slows down drawing of trapezoids is the extra calculations that Phled45 must do for trapezoids: the trapezoid is converted to a polygon and then the polygon's vertices are converted to screen coordinates.

Impact on Source Code

Table 2 shows the comparison among the source code for Caesar, Phled, and Phled45. Most of the source code remained unchanged from Phled to Phled45. The areas of the code most affected were the routines for box manipulation (rotation and placement), file I/O (reading and writing trapezoids), and painting (clipping and merging for painting and erasing).

Table 2.

APPROXIMATE NUMBER of LINES in SOURCE CODE (including comments)			
	CAESAR	PHLED	PHLED45
Transformations	230	230	670
Paint manipulation	1300	1590	2230
"Wire" entry	0	120	480
Read/write files	1610	1960	2140
Box manipulation	530	590	1720
Miscellaneous paint	170	170	360
"Include" files	380	420	620
Other	11580	15780	15780
Total	15800	20860	24000

SUMMARY

Phled45 successfully implemented the ability to enter shapes containing 45° angles. The user interface is exactly the same as Caesar for Manhattan shapes and is quite similar for 45° shapes.

Less than 650 lines of additional source code were required to implement the basic data base operations for 45° shapes. This was because the trapezoid fit well into the existing data base routines for rectangles.

The performance of Phled45 for a Manhattan-only design is quite close to that of Phled. The penalty for entering 45° shapes is acceptable and is only incurred in areas actually containing trapezoids; the short-cuts for rectangles are used whenever possible.

POSTSCRIPT

A year has passed since the original work of implementing 45° capability in Phled. I now want to look at how my work has stood up to the test of time. I will look at which design decisions turned out to be correct and which decisions could have been improved. In addition, I will discuss enhancements to the original implementation that I have made.

The editor forms the foundation for an entire set of physical design tools. In addition to myself, there are now two other software engineers working directly with Phled45. The source code currently totals almost 40,000 lines, and new features are still being added.

Design Decisions

Good Decisions. The effort expended in minimizing the impact to the source code has been well worth it. Caesar is in wide-spread use and this means that there several sources of enhancements to Caesar. It is a simple matter to incorporate these enhancements in Phled45. For example, modifying a set of database

access routines that were originally designed for Caesar [8] to handle trapezoids took less than one week. (Part of this time was accounted for by the differences in the file formats between Caesar and Phled45.) Also, incorporating a river router into Phled45 is anticipated to be completed in only two weeks.

Questionable Decisions. Although the rotated box user interface has been accepted by Phled45's users, it has presented some difficulties in the implementation.

Once I made the decision to use the rotated box, I then had to decide how the box would be rotated. One approach would have been to rotate the box around a fixed point. (See Figure 12.) As is evident from the illustration, it would be hard for a user to predict where the rotated box would be placed. In addition, even though at the start of the rotation the point being rotated about would be the box's "origin" (the point by which the box is moved) the origin would move around the box. So this approach was not desirable from the users' standpoint. But it would have been simple to implement.

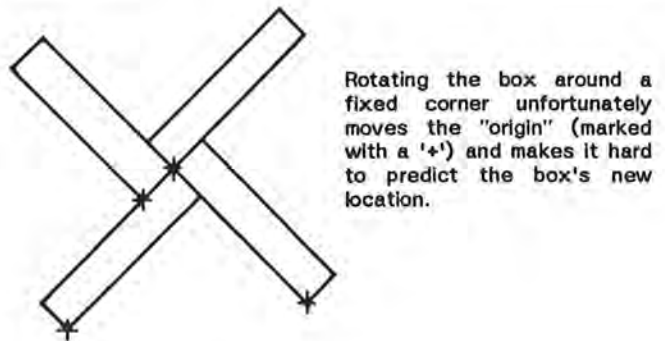


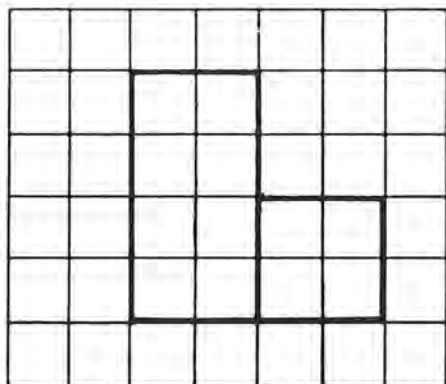
Figure 12. Box Rotation

Because the user interface was the most important consideration for Phled45, I chose to rotate the box about its center. This approach had the advantage that the origin of the box never changed. It was also much easier to predict where the box would lie as it was rotated. But this decision introduced complications into Phled45's source code. First, the center of the box was not necessarily in the editor's coordinate system. But the most serious complication was in calculating where the vertices of the rotated box would lie. Rotation by increments of 45° creates irrational coordinates which then have to be rounded off to lie in the coordinate system. In addition, the editor must ensure that the resulting shape still has 45° edges. The round-off introduced in these calculations resulted in the box "walking" and changing size as it was rotated around. Users complained loudly and often about this behavior, so I was forced to modify the original rotation code to take care of this problem.

The earlier description on clipping described what coordinates were legal to enter in Phled45. I could not allow trapezoids to be placed at the minimum resolution because round-off could occur in erasing

one shape against another. So the question was, how should I align the vertices of a trapezoid so that illegal shapes would not be created yet allow as much of the coordinate system as possible to be available to the user? The answer as shown before was to align the rotated box so that the sum of the x y coordinates is even.

But there was another consideration to aligning the rotated box. Many users want to create designs that have a resolution of say, .01 micron in some parts of their design. But in other places, they may want to create shapes whose dimensions are some multiple of this minimum resolution. Phled provided a means for them to "lock" the Manhattan box into the desired alignment by aligning the corners of the box to a multiple of the minimum resolution. (See Figure 13.)

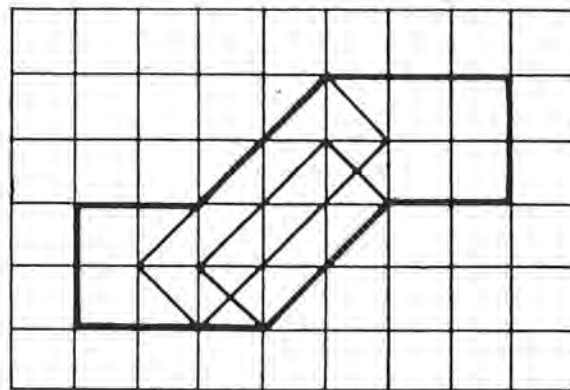


These Manhattan boxes are aligned to multiples of two.

Figure 13. Aligning Manhattan Box

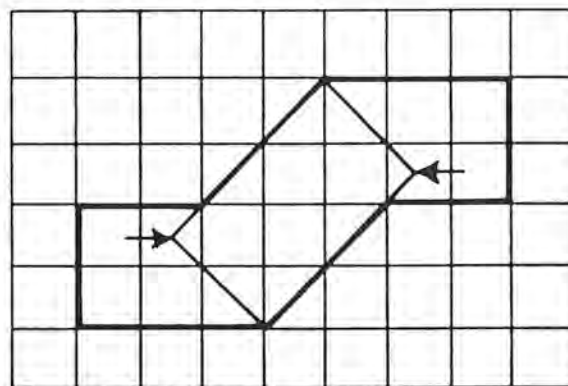
How was the rotated box to be aligned? I wanted to allow the user to easily enter the type of shapes to which he is accustomed: a shape whose vertices are aligned to the user-defined alignment. I could have chosen to align the rotated box's vertices to the alignment, but this would have made it awkward to enter 45° edges. (See Figure 14.) So I decided to align the vertices to half of the user's alignment, with the sum of the vertices being aligned to this alignment. This method was consistent with the decision discussed above to align the sum of the vertices to be even. (In this case, the alignment is 2.) It also allowed the user to easily create his familiar shapes.

Unfortunately, this decision has caused great confusion to the users. Their attitude is, "If I said I only want the box to create shapes every 10 units, why is the rotated box on some 5 unit increments?". I decided to change Phled45 so the rotated box's vertices are aligned to the user's alignment. The objection to the awkward way of entering some 45° shapes has been removed because the vertices of the final shape can now be directly specified by a new extension to the user interface: the polygonal "box".



1 grid = alignment (user-defined multiple of minimum resolution)

If each vertex of the rotated box is aligned to the user-defined alignment, several boxes may be required to enter a 45 degree edge.



1 grid = alignment (user-defined multiple of minimum resolution)

Even though the vertices of the final shape may be aligned to the user-defined alignment, the vertices of the rotated box required to create a 45 degree edge may not be so aligned.

Here the box is aligned so that each x or y coordinate is aligned to half the alignment value; the sum of each (x,y) coordinate is aligned to the alignment value.

Figure 14. Rotated Box Alignment

New Features

Polygonal Box.

I attended a talk on an editor similar to Phled45. [10] It also had a box (Manhattan only) as its user interface, but it has been extended to allow a set of rectangles to be manipulated as one object. This talk gave me the idea of the "box" as an arbitrary polygon. (See Figure 15.) The polygonal box is created by digitizing its vertices. (Vertices can be "back-up", allowing primitive editing.) Once created, the entire box can be moved as before by pushing a mouse button. The polygonal cursor is moved relative to its origin: the point with the smallest y coordinate that has the smallest x coordinate. The area under the polygonal box can be painted or erased in just the same manner that the Manhattan or rotated box is

used. The polygonal box can be left unfinished so that the user can pan and zoom the view while creating this box.

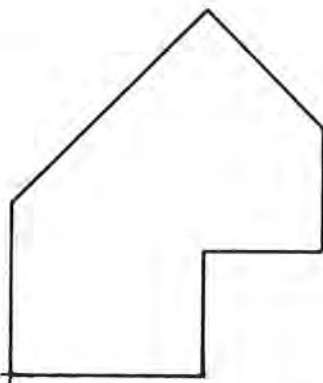


Figure 15. Polygonal "Box"

One objection made by former users of polygon-based layout editors has been that there is no way in Phled45 to enter a polygon. Also, they would like to have the ability to manipulate these polygons once they are entered. Remember, in Phled45, collections of rectangles and trapezoids are used to create areas that appear to be polygons. But there are no actual polygons in the database.

The polygonal box gives Phled45 a user interface that can be closer to a polygon-based editor. Entering shapes proceeds just like digitizing a polygon in an object-based editor. (See Figure 16.) In addition, the polygonal box can select particular parts of the design for editing in one action.

The box data structure was changed to be a polygon when the 45° capability was put into Phled45. This polygon was fractured into a linked list of trapezoids when shapes were to be added or deleted from the database. This fracturing was quite simple because the polygon was always a rotated box.

Modifying Phled45's code to handle the polygonal box was straight-forward. The code for fracturing a general polygon already existed for the Metheus tools, and it was a simple matter to integrate this code into Phled45. I added a new command to create the polygonal cursor by digitizing its vertices. There is also a command to turn the polygonal box back into a Manhattan box.

ACKNOWLEDGEMENTS

I would like to recognize the contributions of John Ousterhout in the design and implementation of the data base algorithms for trapezoids. Barry Roitblat, Howard Landman, and other colleagues have provided technical assistance for the design of Phled45 and this paper. This work was supported by Metheus Corporation. In addition, this project will fulfill part of the requirements for a MSCS at Oregon State University. Finally, I want to thank Drs. D. Sandberg and V.M. Powers at OSU for their assistance.

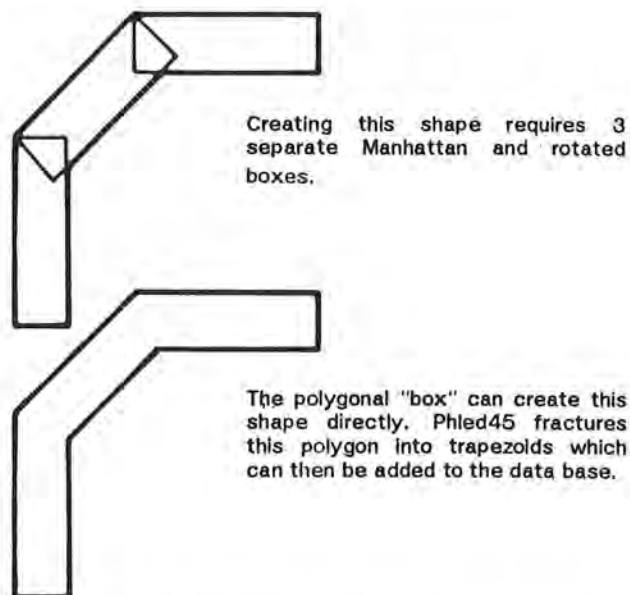


Figure 16. Adding Shapes With Polygonal "Box"

REFERENCES

- [1] Ousterhout, J.K., "Caesar: An Interactive Editor for VLSI Layout", *VLSI Design*, Fourth Quarter 1981, pp. 34-38.
- [2] -----, "Editing VLSI Circuits with Caesar" (User Manual), EECS, University of California, Berkeley.
- [3] -----, "The User Interface and Implementation of Caesar", (to appear) *IEEE Transactions on CAD/ICAS*, July, 1984.
- [4] Metheus Corporation, "Physical Design Tools Manual", 1984.
- [5] Robson, G.D., "Selecting an IC Layout CAD System", *VLSI Design*, March 1984, pp. 28-33.
- [6] Jackson, J.H., "Dynamic Scan-Converted Images with a Frame Buffer Display Device", *Computer Graphics*, Vol 14, No 3, July 1980, ACM.
- [7] Newell, M.E., and Sequin, C.H., "The Inside Story on Self-Intersecting Polygons", *Lambda* (now *VLSI Design*), Second Quarter 1980, pp. 20-24.
- [8] Mayo, R.N., Ousterhout, J.K., and Scott, W.S., "1983 VLSI Tools: Selected Works by the Original Artists", Tpack, Computer Science Division, EECS Department, University of California at Berkeley, 1983.
- [9] Holla, J., "River Router for the Graphics Editor Caesar", Master's Thesis, OGC TR CS/E 83-006, Oregon Graduate Center, Beaverton Oregon.
- [10] Kronmiller, T. and Lee, D., "The Sphinx IC Implementation Environment", Digest of Technical Papers from 1984 IEEE International Conference on Computer Aided-Design, pp. 230-232.

NAME

phled - create and modify physical layout files

SYNOPSIS

phled [options] *filename*

DESCRIPTION

Phled is a graphics editor for physical layout that edits files in the .phl format. If *filename* is specified and it exists, it is read in by *phled* and can then be edited. Otherwise, an empty file is created for editing. See also the *λ750 VLSI Tools Manual* for complete details on *phled*.

The following options are available:

- b If this option is specified for a bit pad and puck, the movement of the puck will be absolute. This is different from the default mouse movement which is relative.
- c *colormap* Override the default colormap by reading in the colormap from *colormap.map*.
- k *pathlist* Use the specified pathlist for control file (i.e. .tec, .dsl, .mnu, .map) lookups. The pathlist is made up of directory names separated by colons (:). Directories may be specified using the '~' notation and "::" is equivalent to ":", ". The first directory in the pathlist is searched first. A '+' means the existing path so "~foo+" means add ~foo to the current path. "+~foo" means add ~foo to the end of the current path. Otherwise the pathlist will override the default path. (The default path is '~:/lib:~wcs/lib').
- l Place the menu on the left side of the display. This is the default placement.
- m *menu* Override the default menu file by using the menu from *menu.mnu*.
- n Do not use any menu.
- p *pathlist* Use the specified pathlist for data file (i.e. .phl file) lookups. The pathlist is made up of directory names separated by colons (:). Directories may be specified using the '~' notation and "::" is equivalent to ":", ". The first directory in the pathlist is searched first. A '+' means the existing path so "~foo+" means add ~foo to the current path. "+~foo" means add ~foo to the end of the current path. Otherwise the pathlist will override the default path. (The default path is '~:/lib:~library/default').
- r Place the menu on the right side of the display.
- s *size* Make the menu area width *size* pixels. Default size is 150 pixels. Bounding box in menu file should be less than this size to fit into the menu area.
- t *technology* Override the default technology (nmos) with the technology in *technology.tec*. *Phled* cannot run if a technology is not successfully loaded.
- u Turn off the checking that ensures that brother cell uses have unique instance names. This may speed up the reading of certain files.
- w *lines* Set height of scrolling text window used for output of error

messages and echoing input from keyboard commands. Default is 2 lines.

COMMAND SUMMARY

When *phled* is invoked, a search is made for a *.cadrc* file. This file can contain command lines for the various VLSI Design Tools. An example of an entry for the physical layout editor is:

```
phled -n -m altmenu
```

These command line options are evaluated before any other options that are typed in when the command is invoked. The current directory is first searched for the *.cadrc* file, then "*~/lib*", and finally "*~/wcs/lib*" is searched. The search will stop at the first *.cadrc* file found that contains a line pertaining to the tool. A line with no arguments can be used to override options in other *.cadrc* files (i.e. this prevents looking for another *.cadrc* file.)

Phled will also look for a startup file and execute *phled* commands from that file before beginning the edit session with the user. The startup file is named *.phledrc*. The current directory is first searched, then "*~/lib*", and finally "*~/wcs/lib/phled*" is searched. The search is terminated as soon as a file of that name is found.

Phled accepts commands from both the keyboard and the mouse. Movement in the mouse is tracked by the crosshair cursor. Some commands in *phled* use the box to determine the area of the drawing to be affected, some commands use the location of the crosshair cursor, and some commands use both.

The buttons of the mouse may have different meanings depending upon which area of the screen the mouse is in. Prompts for the buttons are displayed in the upper right of the screen. In the graphics area of the screen, the left mouse button picks up the entire selector box and moves it to a new location and the right mouse button is used for setting the top corner of the selector box. The middle mouse button in the graphics area paints the area under the selector box in the color(s) which lay under the crosshair cursor when the mouse button was pushed. If the middle button is pushed in the menu area, a command from the menu can be selected. The left button in the menu area rotates the selector box 45 degrees to the left and the right button in the menu area rotates the selector box 45 degrees to the right.

There are two categories of commands which can be typed in. Short commands are single characters with no arguments (with the one exception of '!') and long commands are introduced by a colon (':') and may take some arguments. Each long command can be made up of a series of commands separated from each other by a ';' (semi-colon). For example, ":erase;:pop R;v".

Following are the short commands, their internal names, and a brief description:

- | | |
|---|--|
| 2 | %pandown
Pan one pan increment down. |
| 4 | %panleft
Pan one pan increment left. |
| 5 | %viewall
View entire drawing. |
| 6 | %panright
Pan one pan increment right. |

- B** **%panup**
Pan one pan increment up.
- !** **%shell**
Interpret the rest of command line as a command to the shell.
- a** **%yank2**
Yank the information inside of the box from the layers which lie under the crosshair cursor. If no layers are present there, yank "*L P". The default (unnamed) yank buffer is used for the short command. The yank buffer is not preserved once ":editcell" is invoked again.
- c** **%nocelleexpand**
Unexpand the current cell.
- e** **%erase2**
Erase the information inside of the box from the layers which lie under the crosshair cursor. If no layers are present there, erase "*L P". Erased information will be put into the default (unnamed) yank buffer.
- f** **%find2**
Find the cell under the crosshair cursor and make it the current cell. If the cursor has not moved since last time 'f' was entered, find the parent of the current cell.
- g** **%toggle**
Toggle the grid display.
- h** **%boxleft**
Move the box left one lambda unit.
- i** **%increaseselector**
Continue adding vertices to the polygonal box. This can also be used to backup vertices.
- j** **%boxdown**
Move the box down one lambda unit.
- k** **%boxup**
Move the box up one lambda unit.
- l** **%boxright**
Move the box right one lambda unit.
- m** **%middle**
Center the view on the cursor.
- n** **%resume**
Execute next command from source file. Same as long command ":resume".
- o** **%orthogonal**
Create a rectangular box using the bounding box of the polygonal box.
- p** **%polygon**
Digitize the vertices forming the polygonal box.
- r** **%redraw**
Redraw the graphics portion of the screen.

s	%put2 Put (stuff) the layers which lie under the cursor from the default (unnamed) yank buffer. If no layers are present there, put "*L P S".
u	%undo Undo the effects of the last data modification command. A 'u' can be undone. It is not possible to undo a modification in a subedit after that subedit has been exited. It is also not possible to undo a modification once a new file has been edited.
v	%viewbox View the information inside of the box so that it fills up the graphics area of the screen.
w	%locatelayers Report the layers ("what layers") lying under the cursor.
x	%noareaexpand Unexpand the cells lying under the box.
z	%zoomin Zoom in about the cursor the zoom factor percentage.
C	%cellexpand Expand the current cell to show its paint.
X	%areaexpand Expand the cells lying under the box to show their paint.
Z	%zoomout Zoom out about the cursor the zoom factor percentage.
(dot)	none Repeat the last long command.

Following is a list of the long commands with some common arguments, their internal names and a brief description. The long commands can be shortened to any unambiguous abbreviation. Arguments to long commands can also be abbreviated in this way. The layer names themselves can also be abbreviated.

:array <i>x y</i>	%array Array the current cell <i>x</i> times in the x-direction and <i>y</i> times in the y-direction. Use the current box size as spacing between the cells.
:box <i>keyword amount</i>	%box1 Move the box in the specified way the specified number of lambda units. <i>Keywords</i> are: "up", "down", "left", "right", "xbot", "ybot", "xtop", "ytop". <i>Amount</i> can be positive or negative. The box is aligned to the current alignment.
:changelayer <i>source destination</i>	%changelayer Copy the shapes on the source layer under the box to the destination layer; delete source layer shapes.
:cif [-b -x -a -s <i>scale</i> -o <i>file</i> -l <i>layerlist</i>]	%cif Convert the file being edited from ".phl" format to CIF format. The switches are:

s: The scale is the number of centimicrons per lambda. The default value is 200 (1 lambda is 2 microns).

b: Output a bounding box and text for each cell use. The default is to not output this box.

x: Do not automatically expand the cell uses in the edit cell. Mask geometries will appear only for those cells which are currently expanded. The default is to output CIF for all cells, expanded or not.

a: Do not output arrays as arrays: break each array up into its individual cell placements. The default is to preserve arrays.

o: Output to *file.cif*. The default is to write to *name.cif* where *name* is the file being edited.

l: Convert only the specified layers to CIF. The default layers are "* L P". This option *must* be the last option specified to the cif command.

:cload *file* **%cload**
Load the colormap from *file.map*.

:colormap [?] [* | #] *layerlist* [*r g b*]
%colormap
Set the indicated layer or layers to the specified (decimal) color. If '?' is specified, the red, green and blue values are ignored and instead the color is read out for the layer combination. If '#' is specified, the layer is not treated as a layer but is considered to be an index into the colormap. (See map(5) for details about the colormap.) If '*' is specified, report or change the color for all combinations of the specified layers. (This does not make sense to do for opaque layers nor for colormap indices.)

:connect *layer width*
%connect
This command simulates digitizing a wire with width by allowing a series of constant width shapes to be entered simply by indicating the centerline. After the command is typed in, the user is prompted to move the crosshair cursor and press the right mouse button for each vertex of the wire. Pressing the center button will complete the wire. The left button is used for backing up one vertex. If the width is not given, the last width specified for the layer (either by ":set connect" or ":connect *layer width*") will be used. Similarly, if the layer is not given, the last layer specified along with its width will be used. Once the wire has been entered, its shapes are indistinguishable from any entered by painting. There is an option, set by ":set connect extend", that controls whether or not the paint is extended past the endpoints of the wire. It is also possible to set an option that restricts wires to only horizontal and vertical.

:copycell *id* **%copycell**
Copy the current cell so that its lower left corner is coincident with the lower left corner of the box. An optional *id* may be specified, otherwise a unique *id* is generated from the cell name.

:csave *file* **%csave**
Save the current colormap into *file.map*.

:deletecell %deletecell

Delete the current cell from the edit file only. No change is made to the disk copy of the cell.

:draw [option]**%draw**

Draw is a primitive command used by the ":connect" command. It is not intended to be for general use. The ":draw" command draws a "wire" using the current lower left corner of the box and the lower left corner of the box the last time ":draw" was executed subject to the current connect option values (layer, width, etc). Since a corner is defined by two sides, and draw only creates one side at a time, a corner is always constructed with an octagon. (At small scales this octagon may degenerate into a rectangle). By using an octagon at each corner, we are assured that no matter what direction the next side takes (90 or 45), the corner will be correct. The *done* option resets the last point used for a connection so that the next time ":draw" is executed, a new "wire" will be started.

:drc [ruleset] %drc

Run a design rule check for the all data under the box, visible or not, using the ruleset in "~wcs/lib/DRC/ruleset.rt". If no ruleset is specified, use the ruleset "~wcs/lib/DRC/tec.rt" where *tec* is the current technology. To be consistent with the batch design rule checker (leo), the root cell must be the cell being edited. If it is not, phled will return to the top-level cell with a warning.

:45drc [ruleset]**%45drc**

Run a 45 degree design rule check for the all data under the box, visible or not, using the ruleset in "~wcs/lib/DRC/ruleset.rt45". If no ruleset is specified, use the ruleset "~wcs/lib/DRC/tec.rt45" where *tec* is the current technology. To be consistent with the batch design rule checker (leo45), the root cell must be the cell being edited. If it is not, phled will return to the top-level cell with a warning.

:echo arguments**%echo**

Echo the arguments onto the command line. Useful for macros or source files.

:editcell [name]**%editcell**

Leave editing the current file and begin editing *file.phl*. If no name is given, edit a new file. A warning message is issued if the previous file has not been written and an opportunity is given to write the file.

:erasepaint [layerlist]**%erasepaint**

Erase the paint on the specified layers from under the box. If no layers are specified, erase "* L P". Erased information will be put into the default (unnamed) yank buffer. This is useful for moving paint and groups of labels, ports and subcells.

- :fill** [*direction layerlist*]
%fill
 Find paint crossing one edge of the box and extend the paint to the other edge of the box. If no layers are specified, attempt to fill "*". *Direction* can be "up", "down", "left", or "right".
- :flatten** **%flatten**
 Replace the current cell with the cells making up its array and add them to the edit cell.
- :flushcell** **%flushcell**
 Load in the data for the current cell from the disk.
- :getcell file** **%getcell**
 Get data for *file.phl*, position the cell at the lower left of the box and make this cell the current cell.
- :gridspacing** **%gridspacing**
 Make the grid spacing the same as the current dimensions of the box and turn on the grid display. The default grid spacing is set to 1 lambda at the start of the editing session.
- :gripe** **%gripe**
 Send suggestions and complaints to the system manager.
- :halt** **%halt**
 Suspend the reading or execution of commands from a source file. Used as a command in a source file.
- :help option** **%help**
 Display help information. The option "release" will give information about the phled release. The option "command" will give information about the specified command from these man pages.
- :killmark usermark** **%killmark**
 Make phled forget the mark defined by ":mark *usermark*".
- :kpath pathlist** **%kpath**
 Set the kpath used to search for control files to *pathlist*. Directories are separated from another in the list by a colon (':'). If the first character in the list is a plus ('+'), append the path list to the current path list. If no argument is given, report the current path.
- :label layer text** **%label**
 Add a label to the specified layer with the specified text. If the layer is omitted, look at the paint under the center of the box in order to decide which layer to use.
- :lowerleft** **%lowerleft**
 Center the view on the lower left of the box.
- :macro name string...** **%macro**
 This command allows the user to create a new command composed of existing commands. The command is invoked by *name*. If *name* already exists, it is redefined. String is a list of existing

commands (as entered from the keyboard) or internal names separated by semicolons. *String* may also contain argument substitution specifications of the form $\$i$ where i is replaced by the relative number (starting at 1) of the argument to be interpolated at that point. Argument specifications may also be the special strings $\$*$ which indicates that all arguments are to be interpolated at that point; and $\$-$ which causes the arguments to not be appended to the end of command string (all arguments are normally appended to the end to the command string for compatibility with earlier versions of the program). Arguments may also reference any of the variables controlled by the `:set` command (e.g. $\$alignment$). An argument specification for an argument which does not exist will be replaced by an empty string. The '\$' should be escaped by a preceding backslash ('\') when it appears in macro definitions. If *string* is unspecified, macro *name* is undefined. *Name* will be a long command if it starts with a colon (':') otherwise it will be a short command (must be single character).

:mark *usermark*

%mark

Remember the current location of the box in *usermark* so that it can be retrieved later by `":pushbox"`, `":popbox"`, or `":view"`. *Usermark* must begin with a lowercase letter.

:movecell

%movecell

Move the current cell so that its lower left corner is at the lower left corner of the box.

:mload *file*

%mload

Replace the current menu with the menu in *file.mnu*. If the new menu cannot be displayed, the old menu is restored.

:msave *file*

%msave

Save the current menu in *file.mnu*. If no file name is given, use the name of the last menu loaded.

:noeditlayers [*layerlist*]

%noeditlayers

Make the specified layers uneditable. If no layers are specified, all layers are made editable. If '?' is specified instead of a layer list, display a popup window showing the currently uneditable layers.

:paint [*layerlist*]

%paint

Paint the box with the specified layers.

:pan *direction* [*amount* [*units*]]

%pan1

Pan the view in the specified direction by the specified amount. *Direction* can be "left", "right", "top" or "down". If *units* is omitted, amount is number of pan percentages to move. If *units* is "lambda", the view will be panned *amount* number of lambda units.

:path *pathlist*

%path

Set the path used to search for data files to *pathlist*. Directories are separated from another in the list by a colon (':'). If the first character in the list is a plus ('+'), append the path list to the current path list. If no argument is given, report the current path.

:peek [*layerlist*]

%peek

Temporarily expand the specified layers in the cell(s) under the box to paint. If no layers are specified, expand "* L P". Paint will disappear next time the view is redrawn.

:popbox [*mark*]

%popbox

If no argument is specified, pop the top of the box stack into the box. If a user mark is specified, copy it into the box. System and absolute marks can also be specified. A system mark is denoted by an upper case letter and these system marks are recognized: 'C' is the bounding box of the current cell, 'E' is the bounding box of the edit cell, 'R' is the bounding box of the root cell, 'V' is the current view, and 'P' is the previous view. An absolute mark consists of four integers (x y width height) which are separated by spaces. The absolute mark is aligned to the current alignment.

:port layer text [*function interchangeability*]

%port

Make a port with the specified text on the designated layer. If the layer is omitted, look at the paint under the center of the box in order to decide which layer to use. The function and interchangeability arguments can be omitted and will default to "undefined" and 0, respectively. Function can be one of four recognized text strings ("undefined", "input", "output", "bidirectional", or a unique abbreviation of these four strings) or can simply be a number.

:post return-string representation

%post

Post an item to the menu. *Return-string* is the command that will be given to *phled* to execute when the item is selected. *Representation* is the text that will be displayed. If spaces are to be part of the return string, invoke this command without any arguments and this function will prompt for each argument separately. Then spaces can be part of the return string or the representation.

:pushbox [*mark*]

%pushbox

If no argument is given, push the current box onto the box stack. If a user mark is specified, copy it into the box. System and absolute marks can also be specified. See also ":popbox" and ":mark".

:put [*layerlist*]

%put1

Put the contents of the named yank buffer on the specified layers so that the lower left of the information is at the lower left of the box. If no name is specified, the default (unnamed) yank buffer is used. If no layers are specified, put "*L P S".

:quit

%quit

Quit from *phled*. A warning message is issued if any modified files from the current edit session have not been written and an opportunity is given to write the files.

:reidentify *instance-name*

%reidentifycell

Change the instance name of the current cell.

:relabel *layer text*

%relabel

Edit an existing label to have the specified characteristics. If the layer argument is omitted, retain the current label layer. If only the text argument is given, only the text for the label is changed. If both arguments are omitted, prompt the user for the information.

:rename *file* [*id*]

%rename

Replace the current cell, with the cell *file.phl* at the lower left of the box and make this cell the current cell, with the same orientation and same rows and columns in array (if any).

:report *layer text* [*function interchangeability*]

%report

Edit an existing port to have the specified characteristics. If the layer is not given, retain the current one. If only the text argument is given, only the text for the port is changed. If no arguments are given, the user is prompted for the appropriate responses.

:reset

%reset

Reset the graphics terminal and redisplay the entire screen including text, menu area and graphics. Reload the colormap.

:resume

%resume

Resume reading and executing commands from the source file.

:return

%return

Return from the current subedit.

:rotate [*amount*] [*"buffername"*]

%rotate

Rotate the current cell by *amount* degrees. If the amount is positive or omitted, the direction of rotation is counter-clockwise. Negative amounts will rotate clockwise. The amount is rounded off to a multiple of 90 degrees. If a *buffername* is present, the named yank buffer will be rotated instead of the current cell. If only a " is present, the unnamed yank buffer is used.

:saveall

%saveall

The user is prompted to write or not write each file that has been edited during the current edit session.

:search *regexp*

%search

Search the data under the box to find a label, port or subcell name that matches the regular expression. (See *regex(3)* for information on regular expressions.) The box stack is first cleared. Each matching label, port or bounding box of a matching subcell is

pushed onto the box stack.

:selector {*left / right*}

%selector

Rotate the selector box 45 degrees to the left or right. (This command is the same as pushing the left or right button in the menu area.)

:set option

%set

Set the specified option to the specified value. If no option is given to set, a popup menu will display the current settings and allow them to be changed. Options include:

align size

Set the alignment factor to *size*. The alignment is set to 1 lambda at the start of the edit session. The smallest alignment possible is .5 lambda. The box position is rounded off also to this alignment. Commands that change the size or location of the selector box will align the box to this alignment.

height value

Set the height of the selector box. The box is aligned to the current alignment.

width value

Set the width of the selector box. The box is aligned to the current alignment.

quiet {*on off*}

Default value is "off". "on" means that the "more" message will not be displayed for multiple messages.

pansize value

Set the percentage of the screen size used by ":pan".

zoomfactor value

Set the percentage for screen size for zooming.

connect

Set information used by ":draw" and ":connect". If ":set connect" is used with no values, a popup menu will display the current connect settings and the user can change the values. Options include:

connect extend {*on off*}

If extend is set to "off", the extension of the paint past the endpoints of the "wire" will not be made. "on" is the default value.

connect layer layer

Set the current connect layer to *layer*.

connect snap {*45 90*}

If snap is set to 90, "wires" made by ":connect" and ":draw" will be snapped to 90 degree edges. 45 is the default value.

connect width *layer value*

Set the connect width of *layer* to *value*. The minimum width is 1 lambda. The width is not aligned to the current alignment.

:showdef *name***%showdef**

Show the definition for the macro *name*. Long commands (including long macros) have ':' (colon) for the first character in their name.

:source *file***%source**

Read a command from *file* and execute it. Continue this until there are no more commands or until the command ":halt" is executed.

:subedit**%subedit**

Begin editing the current cell in context. Subedits may be nested,

:technology *file***%technology**

Load the technology in *file.tec*. If any files have been edited and not yet written, the user is prompted to write them first.

:unpost *representation***%unpost**

Unpost the menu item whose display text is *representation*. If no argument is given, the user is prompted to pick a menu item with a mouse button.

:upperright**%upperright**

Center the view on the upper right of the box.

:view *mark***%view1**

Set the current view to *mark*. The mark may be a user mark, a system mark or an absolute mark. See also ":mark" and ":pop-box".

:visiblelayers [*layerlist*]**%visiblelayers**

Display the specified layers. If no layers are specified, display "* L P". If '?' is specified instead of a layer list, display a popup window showing the currently viewed layers.

:writecell [*file*]**%writecell**

Write *file.phl* to the disk. If the file name is not specified, write to the current edit file.

:xmirror ["*buffername*"]**%xmirror**

Mirror the current cell about the x axis (upside down). If a *buffername* is present, the named yank buffer will be mirrored instead of the current cell.

:yank ["*buffername*"] [*layerlist*]**%yank1**

Make a copy of the paint on the specified layers under the box and put into the named yank buffer. If no name is specified the default (unnamed) yank buffer is used. If no layers are specified,

yank "*" L P". The yank buffer's are not preserved once ":editcell" is invoked again.

:ycell [*file*]

%ycell

If *file* is not specified, this will make the current cell into paint using the following steps: the box is set to the bounding box of the current cell, the cell is expanded, all paint, labels and ports are yanked, the current cell is deleted, and the yanked information is put back into the database. If *file* is specified, the file is read in and positioned at the box. Then the same steps are followed as before. This command collapses the cell hierarchy and should be used advisedly.

:ymirror ["*buffername*"]

%ymirror

Mirror the current cell about the y axis (sideways). If a *buffername* is present, the named yank buffer will be mirrored instead of the current cell.

:ysave ["*buffername*"] *file*

%ysave

Save the named yank buffer as a file named *file.phl*.

:zoom [in | out]

%zoom1

Zoom in (or out) the zoom factor percentage about the center of the graphics screen. This command is intended to provide an alternative to the zooming about the cursor done by the short commands 'z' and 'Z'. Since zooming is a commonly executed operation, most users will want short commands to do this. To substitute the zoom about the center of the screen for the zoom about the cursor, use the macros: ":macro Z :zoom out" and ":macro z :zoom in".

:zz

%zz

Write the file being edited to the current file name and exit the editor.

LAYER SUMMARY

Legal layers and their long and short names are determined from reading the technology file *techfile.tec*. Shortnames are 8 characters or less, cannot contain embedded spaces or tabs, and cannot start with '+', '-', '*', '?' or '#'. A list of these short names can be arguments to many of the commands summarized above. The layer list consists of the shortnames separated from each other by spaces. The names may be shortened to any unambiguous abbreviation. In addition, *phled* uses several built-in pseudo-layers. These pseudo-layers are named by upper case letters and these layers can also be arguments to some of the long commands. For example, 'L' represents all labels. ";visible L" would make only the labels visible.

The following is a summary of the layers in the default technology nmos along with the pseudo-layers:

p polysilicon

d diffusion

m metal

- i implant
- c cut
- o overglass
- b buried_contact
- e errors
- s symbolic
- a acknowledge
- v drcerror
- * all of the above layers
- B background
- L labels
- G grid
- P ports
- S subcells
- X selector box

FILES

The several files used by this utility for control and data input, as well as for output, uses the standard Metheus-CV path library conventions for default search path and extension names. Refer to pathnames(7) and cadrc(7) for more information.

- /wcs/bin/phled
- /wcs/lib/nmos.tec technology file
- /wcs/lib/nmos.dsl display file
- /wcs/lib/nmos.map colormap file
- /wcs/lib/nmos.mnu menu file
- /wcs/lib/phled/help/release* help files explaining phled releases

SEE ALSO

phlplot(1), phlex(1), leo(1), cif2phl(1), phl(5), dsl(5), tec(5), cadrc(7), pathnames(7), map(5), mnu(5).

AUTHOR

University of California at Berkeley
Metheus-CV (Ann Lanfri)

BUGS

Label text outside of a cell boundary may not be erased when the cell is. Redraw the display to see what is actually there.

":fill" does not extend 45 degree shapes properly.

":connect" enters wires by digitizing the centerline and fills out the wire by adding half the width to both sides of the wire. Since the same amount is added to both sides, the width of the shape may be wider than the minimum for 45 degree edges.

":help" does not yet support help for individual commands.

Erasing paint on top of unexpanded cells containing ports may cause the displayed paint in the ports to be erased. Redraw to see the actual display.

The -b command line option does not work.

When putting or undoing ports and labels, sometimes they are drawn twice.

The drawing routines are now faster. In some of the processes not all layers are redrawn. Occasionally the redraw command will need to be used to see the new screen.