Special Purpose Hardware for Design Rule Checking

Larry Seiler
Massachusetts Institute of Technology

Special purpose hardware can significantly increase the speed of integrated circuit design rule checking. The architecture described in this paper uses four custom chips to implement a raster scan DRC algorithm. It allows the use of 45° angles and can be programmed to check a wide variety of design rules involving an arbitrary number of layers. A shrink/expand operation allows the use of rasterization grids that are small relative to the minimum feature size. Using the Mead/Conway NMOS design rules[8] and assuming a grid size of 1/2λ, or 1/4 the minimum transistor width, this hardware can completely check a 3000λx3000λ layout in under a minute, if the input data can be provided quickly enough.

1. Introduction

One of the most computationally difficult aspects of integrated circuit design is the problem of checking for design rule violations. Design rules define the ways in which the features on the various layout masks may be positioned with respect to each other. In industrial applications, the problem is usually solved by running design rule checks as batch jobs on large machines. In university applications, the most common approach is to simplify the problem by using less complex design rules and disallowing nonorthogonal angles. Neither approach is completely satisfactory. What is needed is a method of design rule checking that allows the greater complexity of industrial design rules while retaining the speed and simplicity of the university design rule checkers.

Special purpose hardware is one way of satisfying these conflicting requirements. This hardware should be inexpensive enough that it can be included in individual color graphic designer workstations. It should be programmable for wide variety of design rules. It should be extensible to large numbers of layers and should be applicable to hierarchical design rule checking algorithms. Also, it should be able to handle 45° angles. Most industrial designs include 45° angles because they can result in a significant reduction in the area required by a layout. Allowing arbitrary angles provides only a small increase in packing density. The only significant use of angles which are not a multiple of 45° is in bipolar analog devices where circular wires are used to accurately control transistor ratios. It has been claimed that octogonal wires would be a sufficiently close approximation.[5]

This research was supported in part by United States Air Force Contract AFOSR-F49620-80-C-0073 and the Real Time Systems Group of the MIT Laboratory for Computer Science.

This paper is organized as follows. First, an overview is given of the algorithm used by the proposed design rule check hardware and the architecture that implements it. The next section describes algorithms that perform width checking and feature shrinking operations. A custom chip architecture that implements these algorithms is also described. Next, the boundary check operation is introduced for checking errors that depend on edge conditions. Another custom chip architecture implements this operation. A final section summarizes the work and suggests areas for further research.

2. Design Rule Check Hardware Description

The two main categories of design rule checking algorithms differ according to the types of objects they manipulate. Geometrical design rule checkers perform operations on geometrical objects such as rectangles, wires and polygons. Raster scan design rule checkers divide the layout into a grid of cells, each of which is empty or full on each layer. The simplest raster scan algorithms use a fixed size rectangular grid of cells, although variable sized cells and trapezoidal cells are also used.

The design rule check hardware described in this paper implements a fixed grid raster scan algorithm. This algorithm is especially well suited to hardware implementation because the data representation and the operations on the data are very simple. Also, the raster scan format includes local connectivity information, making expensive intersection tests unnecessary. This section starts by describing the basic algorithm, including its relation to hierarchical design rule checking. Next, the hardware architecture that impelments it is described and the components of the architecture are discussed in greater detail. Finally, an estimate is developed for the speed performance of the architecture.

2.1 Top Level Algorithm

The basic structure of the algorithm is similar to one used in a software design rule checker written by Clark Baker.[1] Since IC layouts are usually described by a hierarchical structure of geometrical objects, the first step is to instantiate the hierarchy and create a raster image of the layout. Design rule checks are performed by moving a small rectangular window over each position in the rasterized layout, checking to see if the pattern in the window is valid at each position. For example, a 3x3 window could be used to find all places where a mask is only one unit wide. The pattern matching operations that are performed at each position are called local area design rule checks. The final step in the algorithm involves reporting the positions at which errors were found.

There are two main categories of local area design rule checks. The first is width and spacing checks, which are the most common operations. Shrink and expand operations on masks are also used. They are closely related to width checking. The other main category is general boundary checks. These window operations check the relationship between edges of features on two masks. For example, in the Mead/Conway design rules,[8] polysilicon and diffusion are only permitted to have coincident edges where they form a butting contact.

Local area design rule checks are not suitable for design rules that depend on mask features far away from the position being checked. Rules for pad size or spacing would require impossibly large windows. Rules that depend on the connectivity of the layout cannot in general be checked using windows, because there is no upper limit on the necessary window size. The algorithm will report all potential errors that depend on the connectivity or intended functionality of the layout so that a postprocessor can filter out the spurious errors from those that are genuine. Ideally, the postprocessor would include a node analysis step which would not only filter out potential connectivity errors but would also compare the intended circuit against the circuit that was actually implemented.[7]

What is commonly referred to as hierarchical design rule checking is not so much a method of performing design rule checks as it is a method of reducing the work required by a nonhierarchical algorithm. It is important to consider whether the basic algorithm defined above can be used in a hierarchical design rule check algorithm. One such algorithm operates by removing duplicate geometry from the layout.[11] This would not significantly decrease the running time of any raster scan algorithm because the number of geometrical objects in the layout is reduced rather than the area that the layout occupies. However, it is possible to design a hierarchical algorithm that does reduce area. An example is an algorithm that checks a primitive cell in the usual way and checks a nonprimitive cell by checking its subcells and then checking the areas where its subcells overlap. In any case, hierarchical design rule checking can be done using raster scan algorithms as well as geometric algorithms.

2.2 Top Level Architecture

The design rule check architecture described below implements the inner loop of the above algorithm. Custom chips implement primitive operations involving rasterization and design rule checking. Standard MSI and memory parts complete the system. The resulting hardware will be able to fit on a single PC board and will be inexpensive enough to include as a part of individual color graphic designer workstations. First, the basic architecture is described along with the operations it performs. Next, the interface between the portions of the algorithm implemented in hardware and software is discussed. The extensibility of this architecture to larger layouts and more complex design rules is also considered.

Performing a design rule check with the proposed hardware requires three distinct operations, as illustrated in figure 1. These three operations are performed for each scan line. First the design must be converted into a raster image. This is done by feeding mask features that intersect the current scan line into custom chips in the rasterization unit. The resulting line of rasterized mask data is given to a unit which performs local area design rule checks. This unit performs boolean operations on the rasterized input masks, buffers previous raster lines of the derived masks, and feeds them into two kinds of custom chips that perform primitive DRC operations. The output from the local area DRC unit consists of parallel streams of error bits, where a one indicates an error at that position in the layout. The error reporting hardware converts this into a sequence of error coordinates which are read by the controlling processor. Assuming that the incidence of errors is low, this will result in a significant compression of the error information.

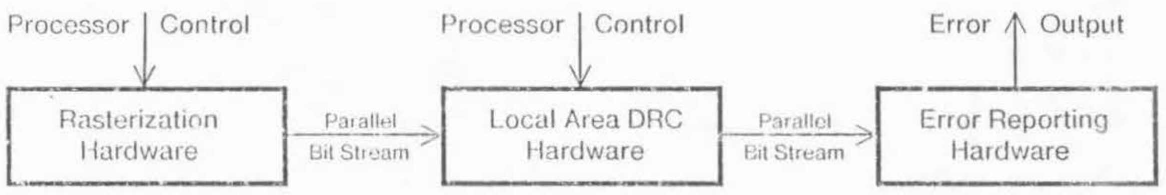


Figure 1: Design Rule Check Hardware System

The hardware architecture in figure 1 cannot stand alone, but must be a part of a design rule checking system. This system contains a controlling processor which includes software to program the local area design rule check unit, convert the layout into the proper format for the rasterization unit, and convert the error reports into human readable output. The design rule check hardware simply speeds up the inner loop of a software algorithm. The basis for choosing these interface points between hardware and software involves the volume of data which is manipulated and the complexity of operations on that data. Software routines can do general data manipulation operations but are relatively slow at processing large amounts of data. On the other hand, data can be manipulated very quickly using special purpose hardware, but the hardware cost increases rapidly as the complexity of the operations increases. The input and output of the local area DRC unit is rasterized data streams, which are easy to process using hardware but would be very time consuming to process using software. The data manipulations required to instantiate the layout prepatory to rasterization and to process the error positions for user output are very general and involve much smaller quantities of information. With the hardware software interface shown above, the hardware implements simple data manipulations on large volumes of data and the software implements general operations on smaller volumes of data.

The above architecture can handle a wide range of layout sizes and design rule complexities. The parameters that limit the size and complexity that can be handled are the number of rasterization chips, the number of parallel mask data lines, the size of the line buffers, and the number of custom chips in the local area design rule check unit. None of these factors place a serious limitation on the usefulness of the hardware. A layout which is too wide for the line buffers can be split into parallel strips that can be checked separately. Adjacent strips must overlap by an amount equal to the largest design rule size. Statistical studies permit good estimates to made as to the number of mask features that will intersect a scan line.[4] If there are too many features for the rasterization chips to handle, a smaller line size can be chosen. It is not necessary to check all of the design rules during one pass through the layout, so the local area design rule check hardware need only include enough custom chips to handle each design rule individually. Finally, only those masks that are actually being used need to be sent to the local area check hardware during a given pass through the design, so the number of parallel mask data lines does not need to be as large as the total number of masks.

2.3 Detailed Architecture

This section describes the rasterization, error reporting, and local area design rule check units in greater detail. The architecture of the local area design rule check unit is described in the most detail, with sample bus sizes and chip counts.

The rasterization unit outputs parallel streams of rasterized mask data to the local area DRC unit. Its input consists of the set of intervals on the current scan line that are covered on each mask. For orthogonal rectangles, this requires that the controlling processor add an interval to the set when the current scan line reaches the start of the box and remove it when it reaches the end. Trapezoids with two horizontal sides and two sides at 45° angles can be rasterized by adding an interval to the set when the current scan line reaches the start of the trapezoid, and then incrementing or decrementing the ends of the interval before each successive scan line until the end of the trapezoid is reached. Polygons and wires must be decomposed into trapezoids. The interval rasterization operation is performed by a custom chip, similar to a chip designed by Bart Locanthi.[6]

The error report unit is the interface between the local area DRC unit and the controlling processor. It converts the parallel streams of error bits into a list of positions where errors were discovered. The type of error at each position is also reported. Most of the error bits will be zero, so this will significantly reduce the amount of information passed to the controlling processor. Typically, the error positions will be saved on a disk file for further processing after the design rule check is complete.

The local area design rule check unit accepts parallel streams of mask data bits and produces parallel streams of error bits. It performs primitive DRC functions such as width tests and boundary checks. It also implements mask shrink and expand operations and boolean operations such as mask intersections, unions, negations, and differences. The boolean operations are used to create derived masks such as transistor gate area, which is defined to be the intersection of the polysilicon and diffusion masks. Since the width tests, boundary checks, and shrink/expand operations require looking at the masks through windows of size up to 4x4, buffers are used to save up to three previous lines of each mask. Figure 2 illustrates the architecture of the local area design rule check unit with sample bus sizes and numbers of chips.

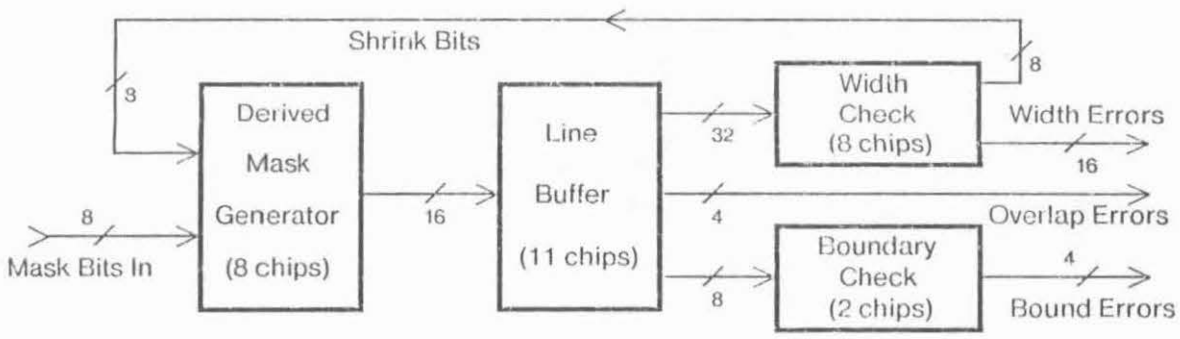


Figure 2: Local Area DRC Hardware

The mask bits input to the the local area DRC unit are fed into the derived mask generator, which uses a custom chip to perform boolean operations on them. The derived masks are fed into the line buffer, which uses standard memory parts to buffer preceding lines. Each custom chip in the width check unit receives a mask from the derived mask generator along with the corresponding bits in the preceding three lines saved in the line buffer. The width check chips each output two width error lines for the input mask and also output the result of a shrink operation on the mask. This operation can also be used for mask expansion. The shrink output is fed back around to the derived mask generator, allowing multiple shrink operations to be done. The boundary check chips each require bits from the current and preceding lines of two derived masks and produce two error outputs. The final group of error lines are output directly from the derived mask generator. These are useful for situations such as overlap tests, where errors are found by subtracting one mask from another. The algorithms and architectures used for the width check and boundary check chips are described in sections 3 and 4, respectively.

2.4 Timing Estimate

At this point, we have enough information to estimate the speed of the design rule check hardware. The basic data rate is determined by the rate at which the rasterizer sequences through the positions on a scan line. Assuming that data is buffered at appropriate places, the most complex operation that must be done during a single cycle is a memory read and write in the line buffers. It is reasonable to assume that this can be done in 200ns. Using a grid size of $1/2\lambda$, and given the hardware configuration in figure 2, the Mead/Conway design rules can be checked in no more than five passes through the design rule checker.[10] Assume that the chip being checked is $3000\lambda \times 3000\lambda$, which is 295 mils on a side if λ equals 2.5 microns. Further assume that there is a 50% overhead resulting from overlapping strips and delays between scan lines. The equation below gives the time needed for a complete design rule check.

2.1 $(3000\lambda)^2 \cdot 200\text{ns}/(1/2\lambda)^2 \cdot 5 \cdot 150\% \quad = \quad 54 \text{ seconds}$

Of course, the controlling processor will not necessarily be able to provide data to the rasterization unit that quickly. Clark Baker's raster scan DRC program takes 49 seconds simply to read the instantiated rectangle file for a chip that size, which contains about 100,000 rectangles.[2] This would have to be done once for each pass through the design rule checker. Experience indicates that it would be much faster to instantiate the chip on the fly from a hierarchical description, rather than read it from a large disk file. Further research must be done to find ways of quickly getting data into the rasterization unit. If software instantiation algorithms are not fast enough, special purpose hardware could be designed to speed that up as well.

3. Width Checking and Feature Shrinking

The most frequent operation in integrated circuit design rules is minimum width checking. Polysilicon, diffusion, metal and contact cut masks all have their own minimum feature size. Spacing checks are simply width checks performed on the complement of a mask, so spacings between features on the same or different masks can be checked in the same way that widths are checked.

This section starts by describing how to check for width errors in mask features which are orthogonal and are small in comparison to the grid size. Then the method is expanded to handle edges at 45° angles. A feature shrinking operation is introduced to allow checking of greater widths. This operation can also be used to expand masks. Next, a custom chip is described that implements these operations. Finally, a notation is developed for specifying width checks and shrink or expand operations.

3.1 Orthogonal Width Checking

Width checks require looking at features in a window which is one greater in size than the width that is being checked. So, a 3x3 window is needed to verify that a mask is at least two units wide and a 4x4 window is needed to check for width three. Figure 3 illustrates the set of patterns that implement these checks. A one indicates a position where the mask is required to be present, a zero indicates a position where the mask is required to not be present, and a dash represents a don't-care position. The mask passes the width 2 test if it matches one of the four orthogonal rotations of one of the patterns in figure 3a. The mask passes the width 3 test if it also matches one of the four orthogonal rotations of one of the patterns in figure 3b.

-	-	-
-	0	-
-	-	-

1	1	-
1	1	-
-	-	0

1	1	-
1	1	-
1	1	-

Figure 3a: Valid 3x3 Windows

-	-	-	-
-	0	-	-
-	-	-	-
-	-	-	-

1	1	1	-
1	1	1	-
1	1	1	-
-	-	-	0

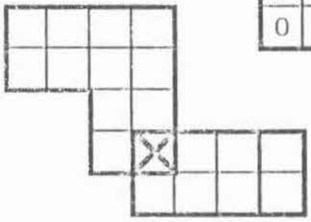
1	1	1	-
1	1	1	-
1	1	1	-
1	1	1	-

Figure 3b: Valid 4x4 Windows

The patterns in figure 3a are correct because a mask fails the width 2 check if and only if it contains a feature of width 1. The patterns in figure 3a check whether the center cell is part of a feature of width 1. If a mask matches the first pattern in figure 3a, then the mask has width zero at this point. If it matches the second pattern, it is part of the corner of an area which is at least 2 units wide. If it matches the third pattern, then it is part of the center or edge of an area which is at least 2 units wide. Therefore, if it does not match any of these patterns, it must be part of a feature which is only one unit wide.

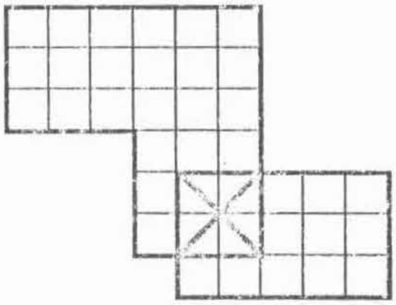
The justification for the patterns in figure 3b is similar. They check whether the center 2x2 box is part of a feature which is greater or less than 2 units wide. If there is a zero in one of the four center cells, then the mask is less than two units wide at that position. If the mask matches the second or third patterns, then the present position is the corner, edge, or interior of an area which is at least three units wide. If none of these patterns are matched, then there must be a feature of width 2 at this position. If a pattern in figure 3a and a pattern in figure 3b are matched, then the mask must be either less than 1 or greater than 2 units wide. This is the test that is needed for the width 3 check.

Figure 4 gives examples of checking for width 2 and width 3. The edges and corners of the rasterized rectangles must fall exactly on the rasterization grid for the width check to work correctly. Rounding is not permitted. Each example contains an error which is marked by an X. The 3x3 and 4x4 windows show the mask pattern around the error positions. It should be noted that if the zero were not present in the corner patterns in figure 3, the errors below would not be detected.



1	1	0
1	1	1
0	1	1

Figure 4a: Width 2 Example



1	1	1	0
1	1	1	1
1	1	1	1
0	1	1	1

Figure 4b: Width 3 Example

There is some question as to whether the pattern in figure 4b is actually an error. The angular distance across the stricture is $2 \cdot (2)^{1/2}$, which will be referred to as 2 diagonal units or 2 diags. This distance is approximately equal to 2.83 orthogonal units. This is only 6% less than the required width of 3 units, and in some fabrication processes it could be considered sufficiently close as to not be an error. To retain the greatest degree of generality it must be possible to select whether or not this case represents a width 3 error.

3.2 Angled Width Checking

Figure 4 gave an example of measuring the width of a feature along a 45° angle rather than orthogonally. Now we will consider how layouts that include 45° angles may be checked for widths 2 and 3. To be checked correctly, all edges must fall on the grid illustrated in figure 5a. Figures 5b and 5c show two more cases that must be allowed in order to do width 2 and 3 checks on a mask with edges at 45° angles.

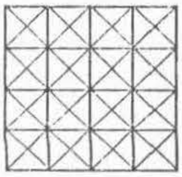


Figure 5a: 45 Degree Grid

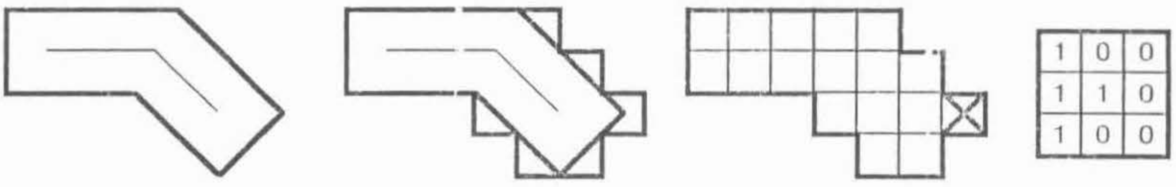
1	0	0
1	1	0
1	0	0

Figure 5b: 3x3 Angled Corner

1	1	0	0
1	1	1	0
1	1	1	0
1	1	0	0

Figure 5c: 4x4 Angled Corner

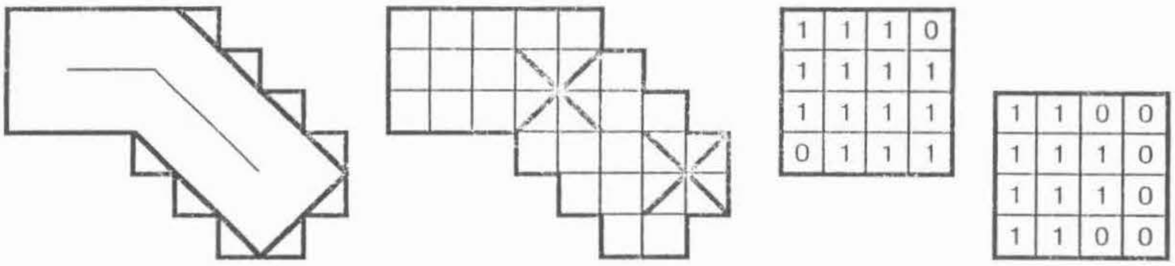
The rules for rasterizing 45° edges are very simple. When a width check is going to be performed, all partially covered raster cells are filled in. When a spacing check is going to be performed, all partially covered cells are marked empty. Figure 6 shows a width check being performed on a wire which contains a 45° bend. When the wire is rasterized, the 45° edges become stairsteps which touch actual edges of the wire at each step. The result is that the rasterized wire is at every point greater than or equal to the correct width, allowing a width check to be done without reporting spurious errors. Since the rasterized wire has the correct width once at each stairstep, any genuine errors will be discovered. The different rasterization rule for spacing checks insures that all spacings are greater than or equal to the correct value. The X in figure 6 marks the position which is illustrated in the window at the right. If the angled corners in figure 5 were not added to the set of valid patterns, this would be reported as an error.



1	0	0
1	1	0
1	0	0

Figure 6: Angled Width 2 Example

Figure 7 gives an example of checking an angled wire of width 3. The width of the angled portion of the wire is 2 diags. or 2.83 units. The upper left X and the upper left window illustrate that this will be detected as a width 3 error unless the pattern in figure 4b has been specified as valid. The lower right X and the lower right window show an example of the 4x4 angled corner from figure 5c.



1	1	1	0
1	1	1	1
1	1	1	1
0	1	1	1

1	1	0	0
1	1	1	0
1	1	1	0
1	1	0	0

Figure 7: Angled Width 3 Example

Allowing the angled corner patterns in figure 5 causes a problem which is illustrated in figure 8. These patterns cannot be distinguished from angled corners. As a result, these errors are not detected if angled corners are allowed. A strong case can be made that these errors are insignificant. However, it is possible to detect these errors and still allow angled corners by doing two width checks. The first check allows angled corners and the second does not. If an error occurs in the second width check that did not occur in the first, and that position is not the corner or edge of an angled box or wire, then it must be one of the errors in figure 8.

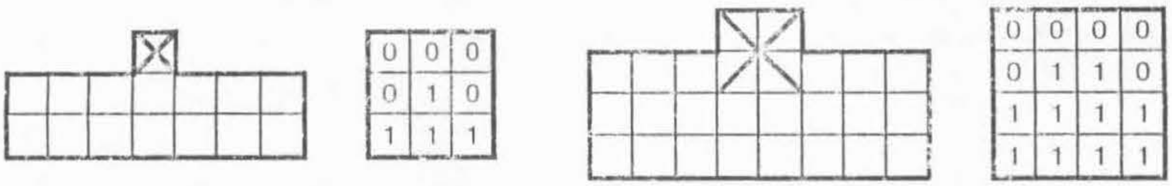


Figure 8: Undetected Errors

The inclusion of 45° edges can result in some undesirable degenerate cases. Figure 9 shows what happens when 45° edges are coincident or within 1/2 diag of each other. Figure 9a illustrates a zero width angled wire which is rasterized for a width check and two angled wires with coincident edges which are rasterized for a spacing check. In the first case, a zero width line causes raster cells to be filled in. In the second case, unwanted holes appear along touching edges. The remedy for these problems is to require the rasterization algorithm to detect coincident 45° edges and either ignore the feature or fill in the raster cell at that point, depending on which of the cases in figure 9a has been detected.

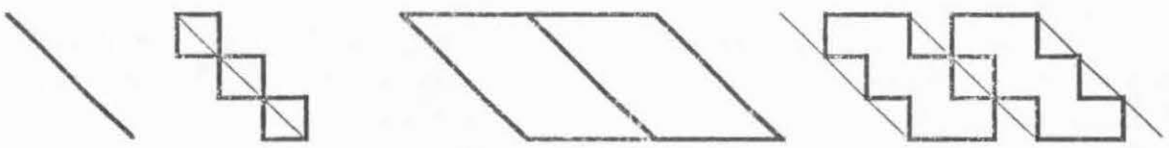


Figure 9a: Coincident Edge Degenerate Cases

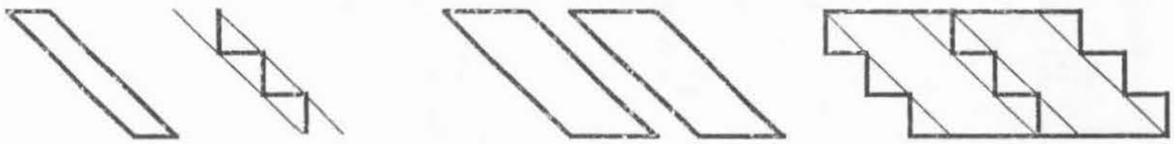


Figure 9b: Near Edge Degenerate Cases

Figure 9b shows what happens when 45° edges fall 1/2 diag apart. In the first case, a wire which is only 1/2 diag wide disappears completely upon a spacing check rasterization. If two angled wires have edges that fall 1/2 diag apart, the gap disappears during a width check rasterization. These problems can be solved by doing both a width 2 check and a spacing 2 check on each mask that may have 45° edges. Checking the structure on the left, for example, will cause a width 2 error to be reported, even though the spacing error that might exist will not be reported. The only complication comes with a mask such as depletion mode implant, which does not have any minumum width or spacing rules. One way to solve this problem is to impose width 2 and spacing 2 rules on this layer and accept the spurious errors that might result. Most of the spurious errors may be filtered out by having the rasterization algorithm report points where 45° edges are 1/2 diag apart. Any legitimate error will be recognized as such by both checks. It should be noted that this is only necessary if 45° edges are allowed in the implant mask. If 45° edges are used for interconnect only and not for transistor or implant areas, then this case will not arise.

Figure 9 also illustrates how the rasterization algorithm deals with acute angles. An inside acute angle will always be detected as a width error and an outside acute angle (an acute angle on the complement of the mask) will always be detected as a spacing error. It would be possible to recognize the occurrance of acute angles and not flag them as errors, but there is little point in doing this. It is not possible to acurately pattern an acute angle onto silicon, so it is not very useful to include one in a layout.

3.3 Feature Shrinking

The algorithms developed above allow mask features to be checked for width 1 errors or width 2 errors using 3x3 and 4x4 windows, respectively. Larger windows could be used to check for greater widths. However, the complexity of the width checks goes up rapidly as the size of the window increases. Also, there is no fixed limit to the widths that will need to be checked. What is needed is a way of making the width check function modular. This can be done by introducing feature shrinking.

The goal of the feature shrink operation is to reduce the size of the features in a mask so that the width 2 or width 3 check can be used. Feature shrinking is done by passing a 3x3 window over the selected mask and producing an output which is one or zero depending whether the mask matches a specified pattern, as for the width 2 check. The difference is that the output is used as another rasterized mask rather than as an error indication. Figure 10 illustrates the four patterns that are used. The orthogonal shrink and angled shrink patterns are discussed below. The shrink/expand and null shrink patterns are discussed at the end of this section. Note that any of the four rotations of the shrink/expand pattern constitute a match.

1	1	1
1	1	1
1	1	1

orthog shrink

-	1	-
1	1	1
-	1	-

angled shrink

1	1	-
1	1	-
-	-	-

shrink/expand

-	-	-
-	1	-
-	-	-

null shrink

Figure 10: Patterns for Feature Shrink

If the orthogonal shrink pattern is applied to a mask, the resulting output will be a one whenever the center of the pattern is in the inside of a feature of width 3 or more. If it is on the edge or corner of a mask feature or is outside, the output will be zero. The result of this is a mask which is the same as the input mask except that all of its horizontal and vertical edges have receded by one unit and all of its angled edges have receded by one diag. Any parts of the mask that are less than three units wide will disappear completely. The angled shrink is almost the same, except that it causes 45° edges to recede by only 1/2 diag. This is illustrated in figure 11. The orthogonal portion of the wire starts out 4 units wide and the 45° portion starts out 3 diags wide. After either shrink, the orthogonal portion of the wire is 2 units wide. After the orthogonal shrink the angled portion is only 1 diag wide, which is a width 2 error, but after the angled shrink it it 2 diags wide. The name of the angled shrink pattern refers to the fact that it does not reduce angular widths as much as the orthogonal shrink.

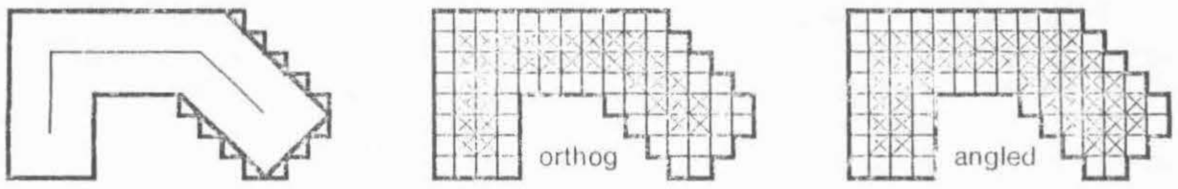


Figure 11: Feature Shrink Example

The local area DRC architecture in figure 2 shows that the shrink outputs from the width check chips are fed back into the derived mask generator for combination with other masks. Therefore, a single mask may be shrunk or expanded as many times as there are width check chips, reducing its width by 2 units each time. This makes it possible to check arbitrarily large widths using only the width 2 and width 3 checks. Since mask features of width less than 3 disappear during the shrink operation, a width 3 check must be performed each time a mask is shrunk. The same patterns that are used to shrink masks can also be used to expand masks, since the expansion of a mask is simply the complement of the shrink of the complement of the mask.

The patterns in figure 12 describe double shrink operations. Shrinking a mask twice using the orthogonal or angled shrink patterns is equivalent to shrinking it once using one of the four double shrink patterns. The two center patterns demonstrate that the order in which a sequence of orthogonal and angled shrinks are done is unimportant because the two shrink operations are commutative. The patterns in figure 12 can also be interpreted as the result of expanding a mask which had a one in the center of the window. Orthogonal expansions produce squares, angled expansions produce angled squares, and a combination of the two produces octagons. This shows that the degree of orthogonal and angular shrinkage or expansion can be selected independently.

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

orthog-orthog

-	1	1	1	-
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
-	1	1	1	-

orthog-angled

-	1	1	1	-
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
-	1	1	1	-

angled-orthog

-	-	1	-	-
-	1	1	1	-
1	1	1	1	1
-	1	1	1	-
-	-	1	-	-

angled-angled

Figure 12: Double Shrink/Expand Patterns

The shrink/expand pattern in figure 10 outputs a one if the cell in the center of the window is part of a feature of size 2x2 or greater. This does not result in any shrinkage of the mask. Instead, it removes features of width 1, as if the mask had been shrunk by half a unit in each dimension and then expanded back again. Figure 13 illustrates using this operation to find the active area of a transistor using the Mead/Conway design rules.[8] The layout on the left depicts a depletion mode pullup transistor combined with a butting contact.

Only the polysilicon and diffusion masks are shown. The center figure illustrates the intersection of these two masks, which includes the unwanted butting contact strip. The shrink/expand operation results in the mask on the right, which is the real transistor gate area. An orthogonal shrink followed by an orthogonal expand can be used to remove areas of width 1 or 2. An angled shrink followed by an angled expand removes features of width 1 or 2 and also rounds off orthogonal corners.

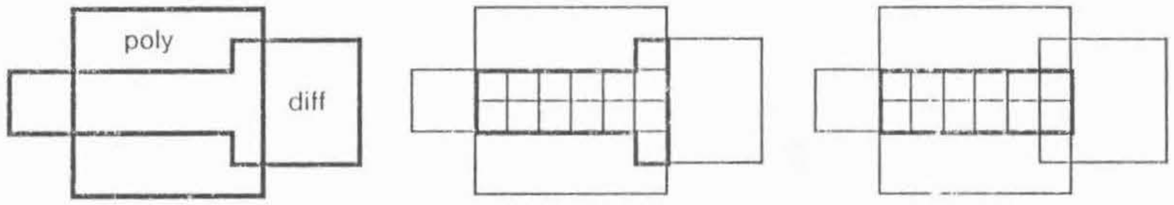


Figure 13: Shrink/Expand Example

Overlap tests can be done by subtracting a shrunk mask from an expanded mask, using the feedback lines into the derived mask generator. However, a shrunk or expanded mask cannot be combined with an unchanged mask. The reason is that the shrink operation displaces the mask that it operates on, since the current mask position is in the corner of the window and the shrink pattern works on the center. The bit entering the derived mask generator from a feedback line is not at the same position on the layout as a bit coming directly from the rasterizer. To combine an unchanged mask with a mask that has been shrunk or expanded, it is necessary to displace it by the same amount as a shrink or expand operation. The null shrink pattern in figure 10 does this. The only change it causes in the input mask is to displace it by the same amount as a shrink or expand. In section 3.4, we will see that there is another use for the null shrink pattern.

3.4 Width Check/Shrink Chip

The previous sections have defined the functionality that is necessary for the width check/shrink chip. Figure 14 gives a structure that implements this functionality. All of the window patterns are impelmented in a clocked PLA which has 44 AND terms. Four successive rows of mask data are input to the chip and the previous three values from each row are saved so that a 4x4 window is input to the PLA. Five control lines are also input to the PLA. Their exact functionality is described below. The output lines WIDTH 1 ERROR and WIDTH 2 ERROR indicate when one or two unit wide mask features are found. SHRINK 1 OUT and SHRINK 2 OUT are the result of applying the shrink operation selected by the control lines. The first applies it to a 3x3 window which uses rows 1–3 and the second applies it to a 3x3 window which uses rows 2–4. This allows the width check chip to output two rows of shrunk mask in parallel. SHRINK 1 OUT can be used as a feedback line to the derived mask generator. A use for SHRINK 2 OUT will be described at the end of this section.

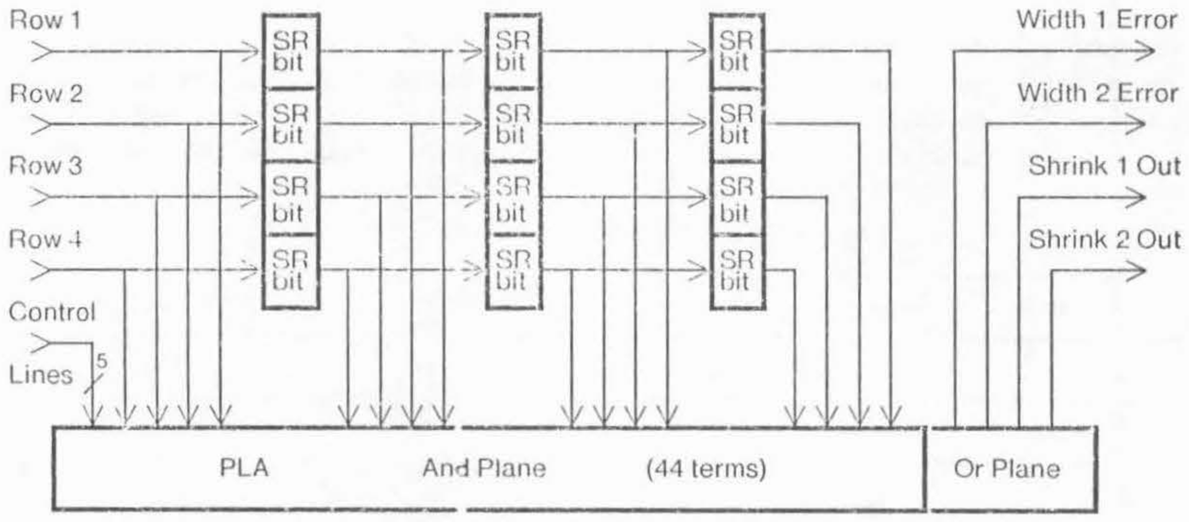


Figure 14: Structure of Width Check/Shrink Chip

Three of the five control lines specify the kind of width check that is performed. These inputs are called WIDTH SELECT, ANGLE OK, and CORNER OK. The WIDTH SELECT line chooses between width 2 and width 3 checking. The WIDTH 2 ERROR output is only nonzero when width 3 checking is selected. This way, the two width error outputs can be OR'ed together externally to produce a single width error signal if the separate information is not needed. The ANGLE OK line controls whether the angled width patterns in figure 4 are treated as width errors. The CORNER OK line causes the angled corners in figure 5 to be accepted. The remaining two control lines specify the pattern that is used to produce the shrink outputs. These inputs are called SHRINK SELECT 1 and SHRINK SELECT 2. Table 1 shows which shrink pattern is used for each combination of the select lines.

Select 1	Select 2	Shrink Type
true	true	orthog shrink
true	false	angled shrink
false	true	shrink/expand
false	false	null shrink

Table 1: Pattern Selection for Feature Shrink

For design rules that contain many large widths and spacings, it would be good to be able to do a double shrink without using feedback lines. Figure 15 shows how several width check/shrink chips could be combined to do double or even triple shrinks at once, reducing the width of mask features by 4 or 6 units at a time. Note that the double shrink requires 6 input rows instead of 4 and the triple shrink requires 8 input rows. Also, a double or triple shrink configuration can be set to do a single or double shrink by specifying a null shrink in the first column of width check chips. All of the control lines in a single column should be tied together so that the same type of shrink is done by all chips in the same column.

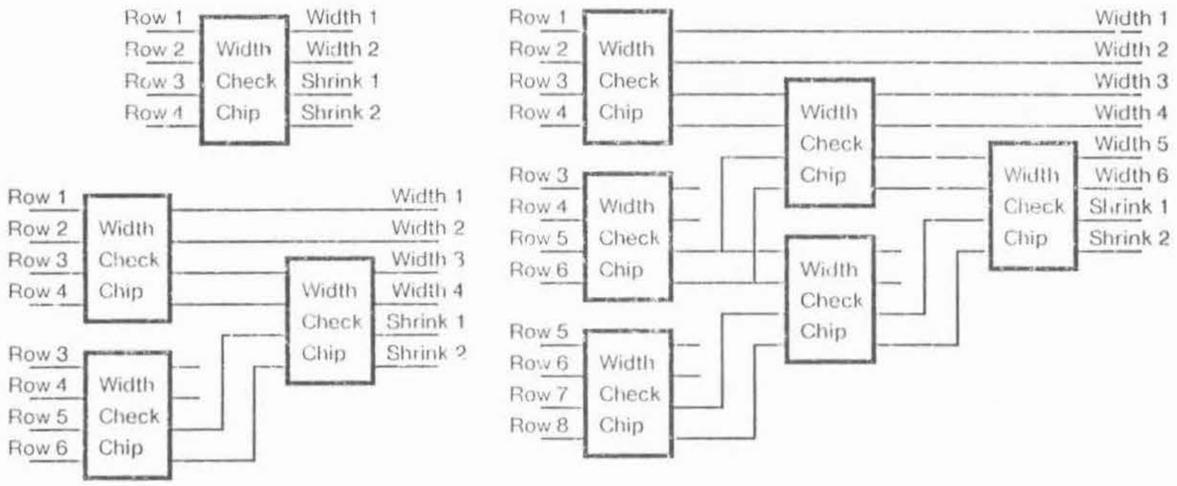


Figure 15: Single, Double, and Triple Width Check Configurations

3.5 Width Check Specification

Now that a custom chip has been defined that can do small width checks and several kinds of feature shrinking, it is necessary to define how larger checks are done. This section shows how to do arbitrarily large width check and feature shrink operations using multiple width check/shrink chips and then defines a notation for specifying width checks and shrink operations.

When a width check is done involving one or more width check/shrink chips, all but the last chip should select width 3 checking. The value of the CORNER OK line should be the same for all of them. ANGLE OK should be false for all except possibly the last one. The number of chips which should have ORTHOG low to select angled shrinking and the state of the ANGLE OK and SELECT lines for the last width check chip depend on the specific width check that is being performed. Table 2 gives the values to use for doing width checks on features up to size 6. A width check is determined by the required orthogonal and angular widths. For example, the sixth line of the table tells how to check for features with a minimum orthogonal width of 4 units and a minimum 45° width of 3.0 diags, or 4.24 units. The table is sufficiently large that the pattern should be clear. The final width check is width 2 if the orthogonal width to be checked is even and width 3 otherwise. Specifying ANGLE OK for the last chip reduces the required diagonal width by 1/2 diag. Changing an orthogonal shrink into an angled shrink reduces the required angled width by 1 diag. This is because the orthogonal shrink causes angled edges to recede by 1 diag while the angled shrink causes them to recede by only 1/2 diag.

orthogonal	angular width		orthogonal	angled	final	final
width	diags	units	shrink	shrink	select	angle ok
2	1.5	2.12	0	0	width 2	no
2	1.0	1.41	0	0	width 2	yes
3	2.5	3.54	0	0	width 3	no
3	2.0	2.83	0	0	width 3	yes
4	3.5	4.95	1	0	width 2	no
4	3.0	4.24	1	0	width 2	yes
4	2.5	3.54	0	1	width 2	no
5	4.5	6.36	1	0	width 3	no
5	4.0	5.66	1	0	width 3	yes
5	3.5	4.95	0	1	width 3	no
6	5.5	7.78	2	0	width 2	no
6	5.0	7.07	2	0	width 2	yes
6	4.5	6.36	1	1	width 2	no
6	4.0	5.66	1	1	width 2	yes

Table 2: Control Line Values for Width Checking

Now it is possible to define a notation for specifying width checks and shrink operations. W2(mask) represents a width 2 check, that is, errors are reported for features of size 1. Subscripts are used to indicate the required diagonal width. For example, $W4_{3.5}$(mask) indicates a check for features with an orthogonal width of 4 units or an angled width of 3.5 diags. The fifth line of table 2 describes how this width check could be achieved. Shrink and expand operations are expressed similarly. S1(mask) describes a shrink operation which causes each edge to recede by one unit. $S3_{2.0}$(mask) denotes a shrink of 3 orthogonal units and 2 diagonal units. This requires one orthogonal shrink and two angled shrink operations. $E1_{0.5}$(mask) specifies an angled mask expansion and is equivalent to $\neg S1_{0.5}(\neg$mask). N1(mask) indicates a single null shrink, which displaces the mask by the same amount as a shrink operation but otherwise leaves it unchanged. SE(mask) indicates a shrink/expand operation, which removes one unit wide features. Finally, a superscript on the mask name indicates the kind of rasterization to do. If NP represents the polysilicon mask, then NP^+ specifies rasterization for a width check and NP^- specifies rasterization for a spacing check.

4. Boundary Checking

So far we have seen how to shrink and expand masks and perform width checks on them. Another type of design rule depends on the relative positions of the edges of two masks. An example of this is the transistor extension rule, which requires polysilicon or diffusion to extend beyond the edges of transistor gate areas. This section starts by giving examples of boundary checks required by the Mead/Conway design rules.[8] Then an architecture is described for a custom chip that implements these checks. Finally, a notation is developed for specifying boundary checks.

4.1 Boundary Check Examples

In order to check design rules which involve boundary interactions, it is necessary to use a 2x2 window to look at two masks simultaneously. Figure 16 shows three examples of boundary errors involving the polysilicon and diffusion masks. For each case, one or more 2x2 windows are shown which identify the presence of the error. A capital P or D indicates that the specified mask must be present at that position, a lowercase letter indicates that the mask must be absent, and a dash represents a don't-care position.

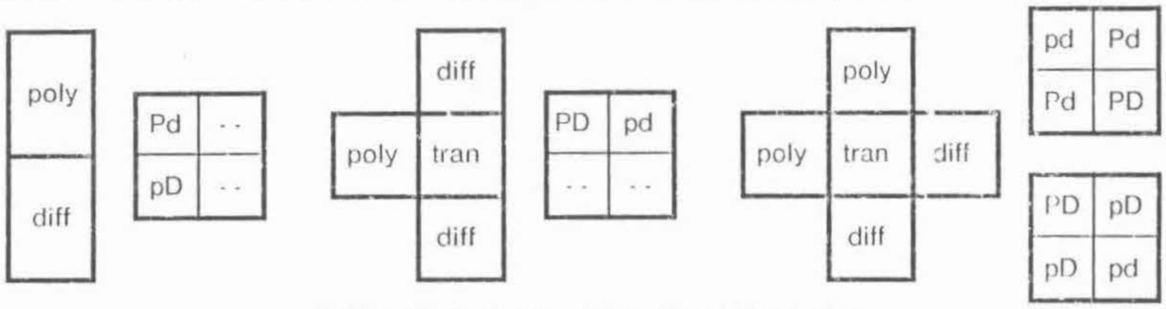


Figure 16: Three Boundary Check Examples

The first boundary error involves the polysilicon/diffusion spacing rule, which requires a 1λ separation except where they cross to form a transistor. This rule cannot be completely checked by a width test on the union of the two masks because they might touch, as shown in the lefthand entry in figure 16. This error occurs whenever a raster cell which contains polysilicon but not diffusion is next to a cell that contains diffusion but not polysilicon. The window that is shown is one of four rotations that must be checked. Note that it is not necessary to check whether the two masks touch at a corner because that case is detected by a width test on the union of the two masks.

The center entry in figure 16 shows a case where the poly mask fails to extend beyond the edge of a transistor gate area. As the window illustrates, this error can be found by looking for a raster cell that contains both polysilicon and diffusion next to a cell that contains neither polysilicon nor diffusion. Again, all four rotations must be checked.

The righthand entry shows a more subtle transistor error. Here, the extension rule is satisfied but the transistor is still incorrect. It is necessary to check that polysilicon and diffusion actually cross the transistor gate area. This can be done by looking at the corners of the transistor. The two righthand windows in figure 16 check for the errors found at the upper left and lower right corners of the transistor gate area. It is necessary to check all four rotations of each window.

4.2 Boundary Check Chip

The structure of the boundary check chip is similar to that of the width check chip in that row inputs are fed through shift registers into a PLA which checks the input window for errors. Unlike width checks and shrink operations, there are too many different possible boundary checks to specify them all in advance. Since it is necessary to be able to program the boundary check chip for the specific check that is desired, a dynamically programmable logic array (DPLA) is used instead of the mask programmable logic array used in the width check/shrink chip. The window patterns programmed into the DPLA are stored in dynamic nodes and must be refreshed periodically.

Figure 17 gives the structure of the boundary check chip. Two rows each from two separate masks are input to the chip. One bit of shift register is provided for each input row so that a 2x2 window on each mask is input to the DPLA. The LOAD SELECT lines control when data on the DATA BUS is used to program the DPLA, which has eight AND terms. The first four terms are OR'ed together to produce ERROR 1 and the other four are OR'ed together to produce ERROR 2. Each of the AND terms in the DPLA specifies a boundary error.

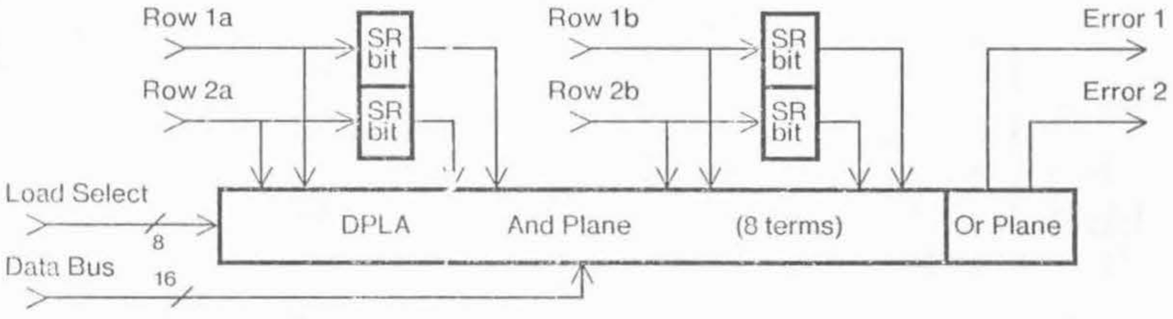


Figure 17: Structure of Boundary Check Chip

Normally, the boundary check chip would input two masks directly from the derived mask generator, but this is not necessary. Figure 18 illustrates an alternate way to use the boundary check chip that allows the inputs to have a shrink operation applied to them before being boundary checked. Note that type of shrink can be selected separately for each width check chip. This is especially useful when the grid size is small relative to the feature size, since many masks will have to be shrunk or expanded before they can be compared.

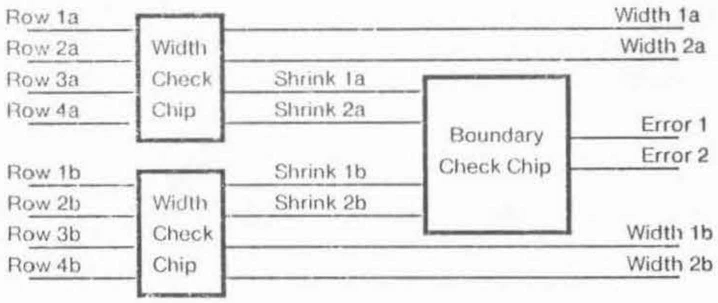


Figure 18: Boundary Check Chip with Pre-shrink

4.3 Boundary Check Notation

A boundary check consists of a set of 2x2 window comparisons performed on two masks. If one of the window patterns is matched, then a boundary error has been found. The two masks are referred to as A and B; they are defined by equations involving shrinks, expands, and boolean operations on masks input from the rasterization unit. Each window is defined by a conjunction of subscripted terms, where each term is a dash or an upper or lower case A or B. The subscript defines the position of each term in the 2x2 window and the chioce of character indicates whether mask A or B is required to be present or absent at that position. For example, if A and B represent the polysilicon and diffusion masks respectively, the leftmost window in figure 16 would be represented as $A_{11}b_{11}a_{21}B_{21}$.

When specifying a boundary check, it is usually necessary to check for several rotations or reflections of the basic window pattern. In the examples in figure 16, each window is one of four orthogonal rotations that must be checked. For clarity, a sequence of windows that are rotations or reflections of each other can be specified as 4[window] to indicate that the four rotations of the window should be checked or 8[window] to indicate that all eight rotations and reflections should be checked. The equations below use this notation to specify the three boundary checks in figure 16. Equation 4.1a is the fully expanded form of equation 4.1.

4.1 $A = NP, B = ND$: $4[A_{11}b_{11}a_{21}B_{21}]$; poly/diffusion spacing

4.1a $A = NP, B = ND$: $A_{11}b_{11}a_{21}B_{21}, A_{12}b_{12}a_{11}B_{11}, A_{22}b_{22}a_{12}B_{12}, A_{21}b_{21}a_{22}B_{22}$; poly/diffusion spacing

4.2 $A = NP, B = ND$: $4[A_{11}B_{11}a_{12}b_{12}]$; transistor edge extension

4.3 $A = NP, B = ND$: $4[a_{11}b_{11}A_{12}b_{12}A_{21}b_{21}A_{22}B_{22}], 4[A_{11}B_{11}a_{12}B_{12}a_{21}B_{21}a_{22}b_{22}]$; transistor corner extension

5. Conclusions

This paper has presented a hardware architecture for a raster scan design rule checking algorithm. The input to the hardware is the set of intervals that are covered on each scan line for each mask. The output is a list of positions where each type of error was found. The hardware can handle a wide variety of design rules, layouts of arbitrary size with an arbitrary number of layers, and features with edges at 45° angles. The use of custom chips makes it small and inexpensive enough to include in individual designer workstations. Furthermore, the hardware is fast enough that the speed of the design rule check is limited by the speed at which the controlling processor can provide input.

Such a significant speed increase can greatly change the way in which design rule checking is used. At present, design rule checking is usually done, if at all, as a postprocessing step after a layout has been mostly completed. In university designs, the design rule check is often not performed at all. The availability of a really fast design rule checker makes it easier to check a layout during all phases of the design effort and not just near the end, when it is hardest to fix any errors that are found.

It should also be noted that the basic architecture of the DRC system is applicable to a variety of other problems. The rasterization unit could be used to drive raster scan printing devices or could be used to scan convert images for a display screen. With slight changes, the architecture could be used to speed up the inner loop of a raster scan node extraction algorithm.[1] By introducing a different set of primitive window operations, the architecture could also be used in image processing applications.

Now that the architecture has been defined, an effort is under way to lay out the four custom chips mentioned in this paper and build a prototype of the design rule check hardware. A paper in preparation will define several variations of the Mead/Conway design rules could be checked using the notations developed in this paper.[10]

Acknowledgements

Many thanks are due to Jon Allen, Clark Baker, Lance Glasser, Gary Kopec, Paul Penfield, and Chris Terman for their ideas and encouragement. Jon Allen deserves special thanks for his support, in all senses of the word.

References

[1] Baker, C.M., Terman, T., "Tools for Verifying Integrated Circuit Designs," *Lambda: the Magazine of VLSI Design*, Volume 1, number 3, Fourth Quarter, 1980

[2] Baker, C.M., Massachusetts Institute of Technology, private communication, December 1980

[3] Baird, H.S., "Fast Algorithms for LSI Artwork Analysis," *Proceedings of the 14th Design Automation Conference*, pp. 303–311, June, 1977

[4] Bently, J.L., Haken, D., Hon, R.W., *Statistics on VLSI Designs,* CMU-CS-80-111, Department of Computer Science, Carnegie-Mellon University, April, 1980

[5] Dunn, W., General Instrument Corporation, private communication, November 1980

[6] Locanthi, B., "Object Oriented Raster Displays," *Proceedings of the Caltech Conference on Very Large Scale Integration,* pp. 215–225, January, 1979

[7] McGrath, E.J., Whitney, T., "Design Integrity and Immunity Checking: A New Look at Layout Verification and Design Rule Checking," *Proceedings of the 17th Design Automation Conference*, Minneapolis, pp. 263–268, June, 1980

[8] Mead, C., Conway, L., *Introduction to VLSI Systems,* Addison-Wesley, Massachusetts, 1980, pp. 47–51

[9] Rosenberg, L.M., "The Evolution of Design Automation to Meet the Challenges of VLSI," *Proceedings of the 17th Design Automation Conference*, Minneapolis, pp. 3–11, June, 1980

[10] Seiler, L., "Formal Definition of the Mead/Conway Design Rules," private communication, MIT VLSI Memo Series (in preparation)

[11] Whitney, T., *A Hierarchical Design Rule Checker,* Master's Thesis, California Institute of Technology (in preparation)