A Physical Design Rule Description

by

Ed McGrath

Technical Report #3236

October 29, 1979

Computer Science Department

California Institute of Technology

Pasadena, California 91125

Silicon Structures Project

sponsored by

Burroughs Corporation, Digital Equipment Corporation,

Hewlett-Packard Company, Honeywell Incorporated,

International Business Machines Corporation,

Intel Corporation, Xerox Corporation,

and the National Science Foundation

Problem:

The current design rule descriptions used throughout the IC design world are couched in English language descriptors such as "width" and "spacing" which are ambiguous in their interpretations and do not accurately reflect the physical limitations of processing.

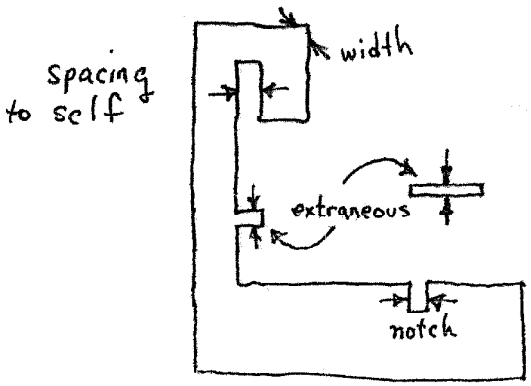
EXAMPLE:

"WIDTH": I know of two commonly used definitions of width, namely:

1.) The minimum distance measured between two non adjacent line segments on a polygon.

2.) A polygon has a minimum width Wmin if a circle of diameter Wmin can pass between all points on the border of the polygon.

Below are illustrated some perfectly good structures for laying out an IC which violate one or both of these definitions.



It should be clear by now that it is not the geometrical properties of the polygons which is important but something else.

I would like to claim that what we're trying to do with these design rules is to make sure that the circuit works. We really don't care if sometimes polygons change shape or short out or even disappear altogether. As long as the circuit works we've done our job. I claim that all design rules boil down to one simple rule:

LAY OUT THE CIRCUIT SUCH THAT UNDER WORST CASE PROCESSING THE CIRCUIT TOPOLOGY IS PRESERVED!

So what I'd like to do is to come up with a design rule description which has the following attributes:

1.) First and foremost, It should accurately and unambiguously describe how to layout a circuit such that even under worst case processing the circuit topology is preserved.

2.) It should also be both teachable and learnable. If it's not teachable it won't be used by schools and if it's not learnable it won't be used by anyone.

3.) It should give some insight into the physical limitations of the process without going into the gory details.

4.) The description itself should not change at all between similar processes and it should change very little from one generation process to the next.

5.) Each rule should be able to be unambiguously expressed in terms of the primitives of a program like the Polygon package.

A design rule description consists of a number of rules to follow when laying out a circuit. If you think about these rules for a while you may discover that each rule is really telling you how to avoid some particular failure mechanism. For example, the minimum width rule tells you how to avoid shorts between adjacent polygons. I believe that to do an intelligent design and even to write an intelligent, accurate checking program you need more information than this. A complete design rule description will give you four pieces of information where current design rule descriptions only give you one or two. These four pieces of information are:

1.) It should tell you what the failure mechanism is you're trying to avoid. eg. shorts, opens, source to drain short, etc. Sometimes design rules tell you this.

2.) It should tell you (without going into gory details) what the worst case processing condition for this failure mechanism is.

3.) It should tell you unambiguously how to avoid this mechanism when laying out the circuit. (This is one thing that current descriptions try to tell you.)

4.) It should describe how to check the layout in terms of the primitives of a program like the Polygon package.

It's a reasonable question to ask, why, if this information is needed, is is not included in design rules today? The answer is that virtually every IC designer today is knowledgable in processing technology and device physics and knows in the back of his mind what these failure mechanisms are and how they happen. There are two problems with this: First, even if every IC designer continues to be a process and physics whiz, the layout designer is not and the guy who writes the checking program probably is not. This means that the layout designer does dumb layouts and the checker writes dumb programs leaving the circuit designer to certify by hand that the layouts are good. Secondly, the IC designer of the future will have to be vertically intesrated more than the designer of today. He will have to be a whiz at logic design, computer architecture, etc. Given that the designer has finite smartness, he will be dumber about processing and physics. The description to follow requires the designer to know no more than is taught in CS 181.

DEFINITIONS

Before I start with the checks themselves I need to make some definitions:

1.) Fixed Bias (FB.layer): This is the sum total of all effects which tend to make the geometry on the silicon bigger (positive bias) or smaller (negative bias) than they were drawn, on the average. Note that statistical variations like overetching are averaged out and are not included here.

All I ask is that you know that such effects exist, I don't wish to burden you with the details of the processing.

If you happen to know a bit about processing, then I'll tell you that some of these effects are things like:

Out diffusion
Field oxide encroachment (bird's beak effect)
Bloating or shrinking done to the data before the masks are made

I'm sure that it will be argued that I should delete this because you should draw what you want and let the processing house handle this. I'd be thrilled if the whole world did this, unfortunately they don't and I include it for that reason. If you happen to be one of those few people who draw what they want on the silicon then you set to use 0 for your fixed bias.

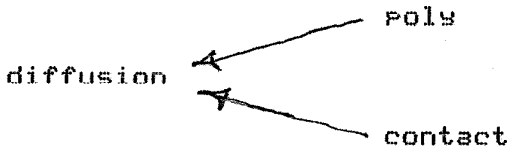
2.) Statistical Bias (SB+(or -).layer): This is the sum total of all statistical variations in the process which tend to make the geometry bigger or smaller than drawn. Some of the processing steps are in fact asymmetric and for this reason I made a distinction between the + bias and the - bias. They can have different values.

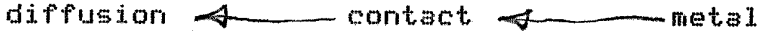
If you happen to know about processing, some of the effects included here are:

Mask dimension tolerances
Over and under etching
Statistical variation in the Fixed Bias

3.) Mask Misalignment (MMi): This is the sum total of all effects which tend to make two different layers not overlay exactly. (i) is the level of indirection in the alignment. Two directly aligned layers have no indirection and have i=0. Two layers aligned through one other layer have 1 level of indirection hence i=1, etc.

eg.



or 

4.) Minimum Final Width (MFW.layer): This is the minimum final width after all processing that is allowable consistent with current density requirements or resistance or whatever.

5.) Minimum Final Spacing (MFS.layer): This is the minimum final spacing after all processing that is allowable consistent with crosstalk requirements or punchthrough or whatever.

6.) Minimum Final Area (MFA.layer): This is the minimum final area after all processing that is allowable consistent with contact resistance constraints or whatever.

Now that these definitions are made it is possible to talk about the design rules in terms of these parameters without actually knowing the values of the numbers. Of course, you need to know the values of the numbers before you do a layout. For industry use, the exact values of these numbers will generally be known ahead of time and layout can begin at any time. For places like Caltech where the designs must work on any number of processes, a technique might be to poll all potential silicon foundries as to the values of these numbers (with the exception of the fixed bias which they would probably consider proprietary anyway) and take the largest of each and design with that. That would guarantee the design working on any of the processes.

DESIGN RULES

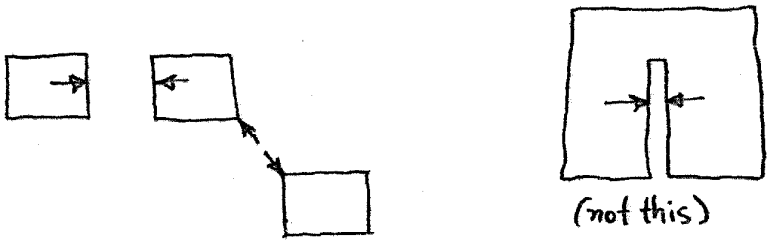
I have not worked out all of the design rules in this scheme yet but a few samples should suffice to show the nature of the description.

Rule: Minimum Drawn Poly Spacing=FB.poly+SBt.poly+MFS.poly
(x2 over SBt.poly; x2 over MFS.poly)

Purpose: To avoid shorts between adjacent poly polygons.

Worst case processing: Poly bigger by SBt.poly

Checking Algorithm: Inflate poly by FB.poly+SBt.poly+$\frac{MFS.poly}{2}$, then check to see that the circuit topology is preserved.



This rule still requires some definitions, but these will be common to some other checks. These definitions are:

Inflate: (sorry, I don't have a good definition yet. All I know is that it is probably not the inflate in the present polygon package.)

Circuit topology: Circuit topology is preserved if the circuit net list is preserved. This is sometimes called connectivity.

Spacing: A circuit meets the minimum width criterion if, when inflated by Smin/2, the circuit topology is preserved.

A point that I sort of skipped over is checking to see that circuit topology is preserved. To do this you generate (unless you are given one by the designer) a circuit net list before you do your inflates or deflates and then generate a new net list after the inflates or deflates and check to see that the net lists are identical. The exact algorithm to do this is not defined yet but it clearly involves device recognition and some knowledge about what a contact does. Such an algorithm is possible since I know of two programs that do just this.

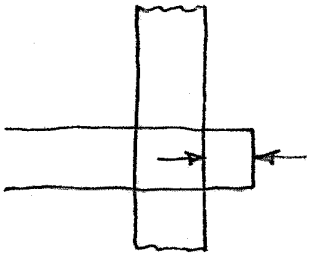
The width check is very similar to the spacing check except that you deflate instead of inflate, so I won't bother describing that one. An interesting check, since it is one which gives traditional checkers a hard time is the gate overlap check.

Rule: Minimum drawn poly overlap of transistor gate
=MM0+FB.diff+SB+.diff+FB.poly+SB-.poly
=___+_____+_____+_____+_____=_____.

Reason: To avoid transistors turning into capacitors (s to d short)

Worst Case Processing: Diffusion bigger by SB, Poly smaller by SB, poly and diffusion misaligned to make overlap smaller.

Checking Algorithm: Inflate diffusion by FB+SB, Inflate poly by FB-SB-, Misalign by MM0 n,s,e,w and check for preservation of circuit topology after each misalignment. (note that if you allow 45 degree lines you need to misalign ne,se,nw,sw also; if you allow all angles you're up the creek without a paddle.)



Note here that I've used a primitive called translate which is not now part of the polygon package.

SUMMARY

What I've tried to do here is to describe a new way to attack the problem of design rules. I'd like to think that the rules are teachable and learnable to and by people who are not processing whizzes. By the same token, it gives some physical significance to the rules and where they come from. I would like to think that the descriptions are explicit enough so that the confusion over the rules that has existed in the past is lessened, and that the checking problem is simplified since the rules will all be described in terms of the primitives of a polygon package like program.