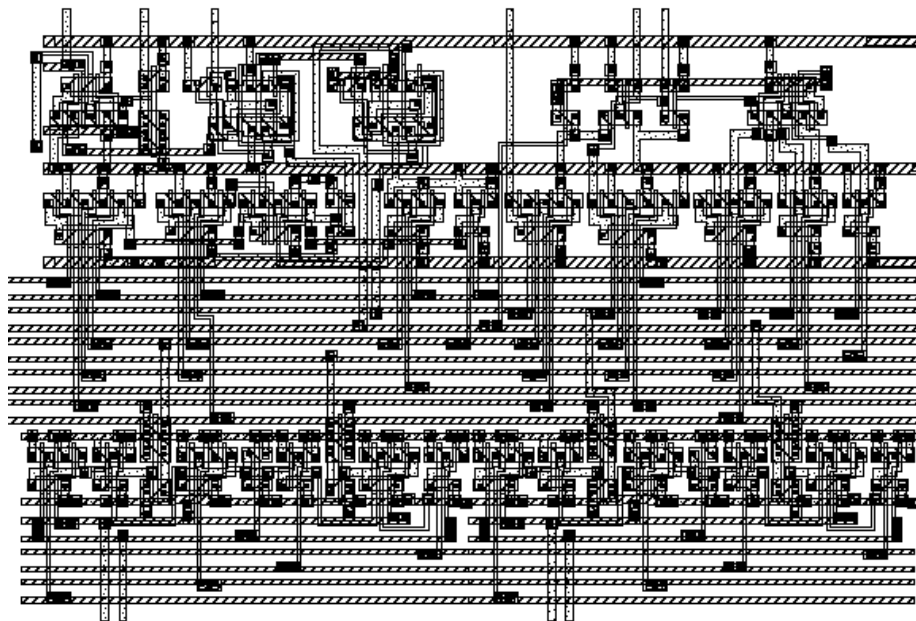


ICED™

DRC-NT Design Rules Checker

Reference Manual



Version 3.xx

IC Editors, Inc.

© 2001 by IC Editors, Inc.

No part of the information contained in this manual may be represented in any form without the prior written consent of IC Editors, Inc.

The software described in this manual is subject to change without notice. Although all information is given in good faith, neither IC Editors, Inc. nor its agents accept any liability for any loss or damage arising from use of the software or from use of any of the information provided herein.

Acknowledgments:

The majority of this manual was written or revised by Ference Professional Services in Sonoma, CA. We are also responsible for formatting the text and creating the screen captures that illustrate the examples.

Michael Gentry of MGC, Inc. created the layout that is used on the cover and as a frontispiece. It is a section of a CMOS simulation of a 74181 4-bit ALU.

Bob Fleming of Aether Wire & Location, Inc. provided a rule set used for parts of the Advanced Tutorial.

Table of Contents

INTRODUCTION	5
Target Audience	6
Manual Organization	7
Other Available Programs	8
GETTING STARTED	9
Program Requirements	10
Installation	11
Quick Tutorial	12
Please read the following section <i>before</i> any technical support calls.	
Troubleshooting.....	27
FUNDAMENTALS OF DESIGN RULE VERIFICATION	31
What Are Design Rules?	32
How Do Design Rule Checkers Work?	39
HOW THE DRC WORKS.....	45
Generating the Input Files and Running the Program.....	47
Looking at the Results	49
Layer Processing.....	55
Spacing Verification	84
Other Verification Rules.....	103
Electrical Connections	110
Panel Processing.....	118
Hierarchical Checking and Hierarchical Output.....	134
Optimizing DRC Runs.....	151
DRC RULES SYNTAX.....	171
General Syntax Restrictions.....	172
2_ONLY DRC version control	176
286_ONLY DRC version control	178
3_ONLY DRC version control	179
ALL_DANGER Prevent cell flattening for dangerous operations.....	180

Table of Contents

ALL_SAFE	Force cell flattening for dangerous operations.....	181
ALLOW_QUICK	Avoid warning prompt for QUICK_PASS processing	182
AND	Boolean AND of two layers.....	183
ASPECT_RATIO	Classify shapes by relative dimensions	184
The Assignment Rule	Copy layer or inverse of layer.....	187
BAD_POLY	Assign layer number for bad polygons.....	189
BLOAT	Expand shapes	190
BLOAT_ANGLE	Define angle for BLOAT rule.....	191
BORDER	Explicitly define panel overlap	193
BOUNDS	Classify shapes by the size of their bounding box	194
BRIDGE	Recognize air bridges.....	196
CONNECT	Electrically connect layers	200
CONST	Define constant value.....	203
CUT_RESOLUTION	Place cut lines on specific grid.....	205
DANGER_CELL	Prevent cell flattening for dangerous operations.....	207
DANGER_LAYER	Override cell flattening for certain layers	209
DETAIL	Turn detailed logging on or off.....	210
HOLE_AREA_FRACTION	Classify polygons with holes.....	211
IN_CELL	Classify shapes in certain cells.....	215
INCLUDE	Allow rules file nesting	216
INPUT_LAYER	Define input layers.....	217
IS_BOX	Classify rectangles by size	222
IS_CIRCLE	Classify polygons with circular shape.....	225
ISLANDS	Find Holes.....	230
MAX_ANGLE	Find sharp points in notches	231
MAX_COUNT	Change maximum number of errors found before warning.....	233
MAX_SPACING	Classify shapes by distance	235
MIN_ANGLE	Find sharp points.....	242
MIN_AREA	Find small shapes	243
MIN_FILL	Verify layer coverage of design area	245
MIN_NOTCH	Find small notches	248
MIN_SIDE	Find shapes with at least one small side.....	251
MIN_SPACING	Find spacing errors	252
MIN_WIDTH	Find shapes with small width	271

Table of Contents

MODIFY LAYER	Define layer used as both an input and output layer	273
NO_CHECK_INPUT	Prevent some bad polygons from being marked	276
NO_HIER_WARNING	Prevent warning during hierarchical output	277
NO_PANELS	Execute DRC on entire design at once.....	278
NO_RUL	Prevent warning when source rules file is missing.....	279
NO_WARN_ACUTE	Prevent marking acute angles.....	280
NOT	Copy inverse of layer	281
OFF_GRID	Find vertices that are not on resolution grid.....	282
OR	Boolean OR of two layers	283
OUTPUT LAYER	Define layer for output.....	284
OVERLAPPING	Find shapes with common area	288
PANEL_VERTICES	Control number of vertices per panel.....	290
PANELX and PANELY	Define maximum panel size.....	293
RULE_SET	Define sets of rules to control execution.....	295
SAFE_CELL	Flatten only certain cells for dangerous operations.....	297
SAFE_LAYER	Force cell flattening for critical layers	299
SCRATCH LAYER	Define temporary layer	300
SHRINK	Shrink shapes uniformly	302
SNAP	Relocate vertices on resolution grid.....	304
SNAP45	Relocate vertices on resolution grid preserving slope of 45° angles	306
STAMP	Electrically connect poor conductors.....	308
STOP_ON_MAX_COUNT	Halt DRC on maximum number of errors.....	310
TOUCHING	Find touching shapes on different layers.....	311
WARN_ACUTE	Assign layer number for acute angle warning marks	313
WIRE_WIDTH	Set error wire width for all error layers.....	315
XOR	Boolean exclusive OR	316
RUNNING THE DRC		317
DRC Rules Compilation		319
Running the DRC		329
DRC Output Files		361

Table of Contents

ADVANCED TUTORIAL	379
Simple Spacing Check	381
Directional Spacing Check	391
Finding Errors Involving Touching Shapes	397
Tests That Involve Electrical Connections	402
Creation of Shapes for Export	418
Hierarchical Output	429
Speeding Long DRC Runs.....	440
Conclusion.....	448
APPENDIX A: OBSOLETE SYNTAX	449
Obsolete DRC Rules.....	449
MAX_QUAD Limited air bridge recognition.....	450
RECTANGLES Find shapes that are not rectangles of specific sizes	451
SKIPPED_POLY Assign layer number for shapes unknown to DRC.....	452
OUTPUT LAYER Obsolete Keywords.....	453
INDEX	455

Introduction

The DRC (**D**esign **R**ules **C**hecker) program from IC Editors, Inc. is a rules-driven program to manipulate layout data and verify technology specific layout restrictions for integrated circuit mask sets. The algorithms used by the DRC allow it to process the data for an entire semiconductor chip on most personal computers at reasonable speeds.

The DRC program combines layer generation algorithms (such as bloats, shrinks, and Boolean operations) with size, spacing, and shape rules to verify that your design meets technology dependant design criteria. The program can also be used to generate mask layers for import back into your design.

Target Audience

There are two different classes of DRC users:

- **Design Rule Writers** These users create DRC rules files and use them on design data and/or testcases.
- **End Users** These users are provided with rules files from another party, but they are responsible for running the program on design data and possibly for installing the program as well.

Design rule writers will be able to do their job best after reading this entire manual. Familiarity with how to execute the program is required in addition to familiarity with the syntax of rules to test rule sets. A thorough understanding of the fundamentals of design rule verification as well as the specifics of how the DRC verifies huge amounts of data with only the memory available on a PC is critical to ensure that all design errors are found by the program.

End users may be able to skip ahead to “Running the DRC” on page 317, but they will be better able to troubleshoot problems if they read everything with the possible exception of the “DRC Rules Syntax” section.

Manual Organization

This manual is organized into the following sections:

If you will not be writing a DRC rule set, you may want to skip ahead to "Running the DRC" after reading the "Getting Started" information.

Recent versions of the layout editor support executing simple DRC operations from inside of the layout editor. However this manual does not specifically cover this use of the DRC.

"Getting Started" covers the program requirements and installation. This section also includes a brief tutorial covering the basic steps for preparing the input files and running the rules compiler and the DRC program.

"Fundamentals of Design Rule Verification" introduces the processes involved in verifying design rules and presents the reasons for some of the more complex features of the DRC. This section is intended primarily for users who have little experience with design rule checkers.

"How the DRC Works" covers the theory behind all DRC features in a detailed manner. Once you have completed this section, you will have at least a basic understanding of how the DRC works. To use the DRC effectively, you will have to read the more detailed syntax sections of features you want to use, but you will have learned enough to know where to look to solve unique problems and avoid common hazards.

"DRC Rules Syntax" covers the syntax for all statements in a DRC rule set. Detailed examples are included. The rules are listed in alphabetical order.

"Running the DRC" describes how to execute the rules compiler and the DRC program after you have a complete rule set. All command line options are covered with examples. The output files are completely described. You will also learn how to import the results of the DRC run into the ICED™ layout editor.

The **"Advanced Tutorial"** uses examples provided on the installation diskettes to take you through all the steps in the verification of a realistic semiconductor design.

"Obsolete Features" mentions some of the rules and command line options that were used by older versions of the program. These features are still supported in the current version so users with older rule sets may still use them without modification.

Other Available Programs

The DRC is intended to be executed in conjunction with the ICED™ layout editor available separately from IC Editors, Inc. This layout editor is required to export the input data file for the DRC from the layout data, and to import the geometry created by the DRC.

The DRC program uses many of the same rules and features as the NLE utility. The NLE utility contains additional rules that allow you to perform device recognition and electrical connection check (ECC) tests to find shorts and opens. The layout netlist generated by the NLE can be compared to a schematic netlist with the LVS utility. (Both the NLE and LVS utilities are available separately from IC Editors, Inc.)

Getting Started

Program Requirements

The DRC program may execute with as little as 8 Megabytes, but run times are likely to be long. We recommend that you execute the DRC program on a computer with at least 16 Megabytes of memory. The DRC does create scratch files for virtual memory, however this disk swapping will slow the program down.

The scratch files created by the DRC can grow very large. We have seen scratch files over 1 Gigabyte in size for large chips. If you have limited memory for the DRC, you will need plenty of free disk space.

You must use version 2.0, or a more recent version, of the ICED™ layout editor to generate the data for the DRC. Recent versions of the layout editor support executing simple DRC operations from inside of the layout editor. However this manual does not specifically cover this use of the DRC. Longer DRC runs should be executed in a DOS console window, outside of the layout editor.

The DRC requires a key on your printer port or USB port for copy protection. IC Editors, Inc. provides this key when you purchase the program. Install the key on the appropriate port. (Connect your printer cable to the key if necessary.) The customized copy protection program DRCnAUX.EXE will look for this key. Do not remove or rename this program. Do not copy it when sending files to other users. You may overwrite their customized copy.

Installation

Follow the instructions provided in the installation software. You will need to specify the name of the ICED™ layout editor installation directory. This can be a new or an existing directory. If you specify a directory that does not already exist, it will be created. You can specify an existing directory from a previous DRC or ICED™ installation. All executable files will be stored here. We refer to this directory throughout this manual as Q:\ICED. Whenever you see "Q:\ICED", replace the 'Q' with the drive letter you chose during installation. Replace "ICED" with the name of the installation directory.

The DRC requires a key on your printer port or USB port.

Some useful examples are included during installation. We will be using some of these files in the tutorial below. See a list of rule file examples on page 26.

The recommended method to launch any of the ICED™ products is to first open a DOS console window with the icon created on your desktop during installation. This icon is labeled "ICED" and displays a representation of a silicon wafer. You can type the DRC and rules compiler command lines at the prompt in console window. The path to the executable files is already added to the system search path when you execute the programs in this manner.

Close the console window by typing EXIT at the prompt or by clicking the 'X' in the upper right corner of the window.

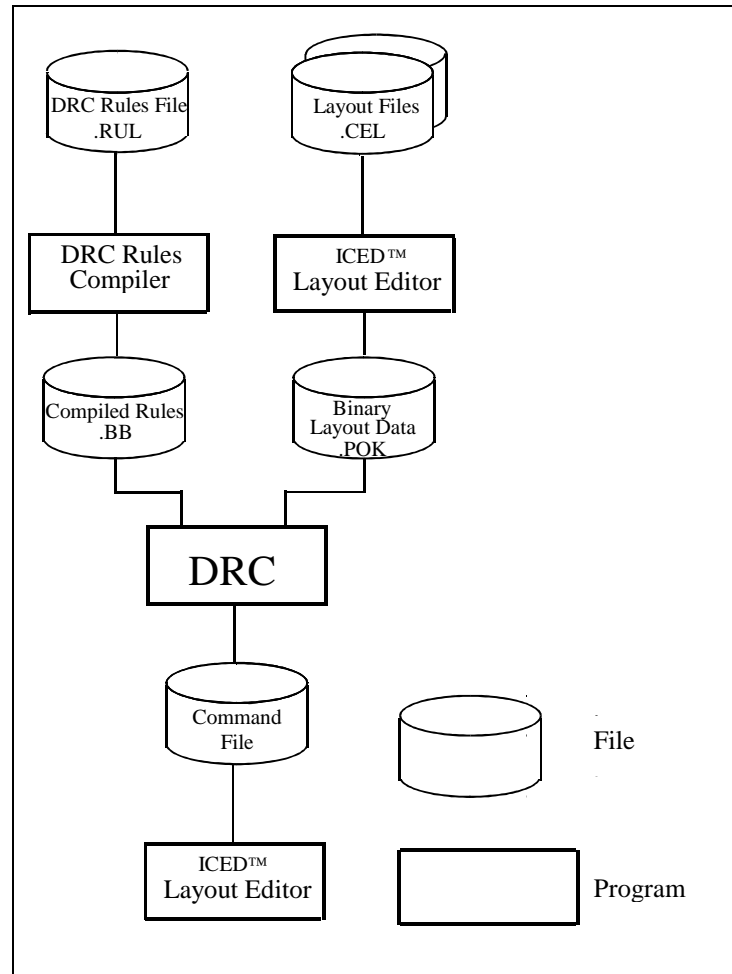
Change the current directory to the desired directory with the DOS command "CD *dir_name*" before executing the DRC program. See how this works in the following tutorial.

Quick Tutorial

This tutorial will cover one of the examples provided with the installation.

The flow of data to prepare the input files is shown in Figure 1.

To run the DRC, you must have two input files. The first contains rules for manipulation of the layers and for design rule testing. These rules must be compiled by the DRC rules compiler prior to executing the DRC. The second file contains layout data generated by the DRC command in the ICED™ layout editor.



The recommended method to launch any of the ICED™ products is to first open a DOS console window with the icon created on your desktop during installation. This icon is

labeled “ICED” and displays a representation of a silicon wafer. Click this icon now using the left button of your mouse.

The executable search path is modified before the console window opens. This means that the operating system will automatically search in the correct place for the DRC programs. The current directory is set to the ICED™ directory, Q:\ICED¹.

Copy the files for this tutorial to a new, empty working directory. This allows you to keep the original copies of the sample files intact for future reference.

Before we create the new directory, we need to make the current DOS drive a drive with some available space. Replace the drive letter "C" in the following command to a disk drive with free space and type at the console prompt:

C:

The exact location and name of the tutorial directory is not important. Create it wherever it is convenient.

Now create the new directory with the command:

MD DRCTUTR

Make this new directory the current directory with the command:

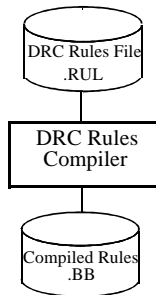
CD DRCTUTR

Now copy the required files with the following commands:

COPY Q:\ICED\SAMPLES\DRC\TRIVIAL.CEL
COPY Q:\ICED\SAMPLES\DRC\EXAMPLE1.RUL

¹ Throughout this manual, Q: and \ICED are used to represent the drive and directory path where you have installed the DRC. If you have installed the software on your C drive in the directory \ICWIN, you should replace Q: with C: and \ICED with \ICWIN.

Preparing the Rules File



The rules file we will be using is **EXAMPLE1.RUL**.

The contents of the file are shown in Figure 2. The lines that are not bolded are comments.

This rules file must be compiled by the DRC rules compiler. The command must be typed at the console prompt. Type the following:

D3RUL-NT² EXAMPLE1

D3RUL-NT.EXE is the name of the program and the rules file is **EXAMPLE1.RUL**. This program will create the compiled rules in a file named **EXAMPLE1.BB**. We will use this file later when we run the DRC.

```

3ONLY! Tells DRC version 2 to ignore next line.
ALL_SAFE
!Will be explained later. For now, use ALL_SAFE in all rule sets.
input layer 1 boxes; 3 one_box;
! ICED layer 1 will be referred to as "boxes" in the following
! rules.
! ICED layer 3 will be referred to as "one_box"
! The semi-colon between layers is required. The semi-colon
! at the end of any line is optional.

output layer 20 too_close1; 20 too_close2;
! Two layers, referred to as "too_close1" and "too_close2" in
! these rules, will be output to ICED layer 20.

DETAIL ON
! This should NOT normally be used. In most real runs, it will
! produce a too-long log file. Use DETAIL ON only for small
! subsets of normal runs.

output layer 20 narrow;
! Layer "narrow" will be output to ICED layer 20.

const min_distance=10;
! The name "min_distance" can now be used instead of 10.

too_close1=minspacing(boxes, boxes, min_distance)
! Check spacing between polygons in layer 1.

too_close2=minspacing(boxes, one_box, 7)
! Check spacing between layer 1 and layer 3.

narrow=minwidth(boxes, min_distance);
! Check width.

badpoly 0
! Bad polygon output is suppressed. ("Bad polygons" will be
! explained later.) A non-zero number would have copied bad
! polygons to an ICED layer.
  
```

Figure 2: EXAMPLE1.RUL

² The executable file for released versions for Windows is D3RUL-NT.EXE. The executable file for Beta Windows versions is named D3RU-NTX.EXE.

The console messages will be very brief. The version of the compiler, along with a copyright notice is followed by a report of how much memory is available to the compiler. When the compiler finds no errors, the next line is:

Done.

This indicates a successful compile.

If you are not already familiar with an ASCII file editor, use the editor that comes with your DOS installation, EDIT.COM.

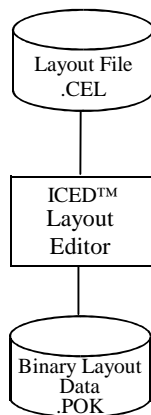
The log file created by the rules compiler is EXAMPLE1.RLO. When the compiler finds an error, a message will be printed both on your screen and in this file. Browse or edit this file now. The version and copyright information is followed by the contents of the source rules file. This is followed by lines similar to Figure 3.

5 layers used.				
	Name	Number	Line	Type
1	BOXES	1	4	INPUT
2	ONE_BOX	3	4	INPUT
3	TOO_CLOSE1	20	11	OUTPUT
4	TOO_CLOSE2	20	11	OUTPUT
5	NARROW	20	20	OUTPUT
1 named constants.				
	Name	Line	Value	
	MIN_DISTANCE	23	10	
5 actions, requiring 3 passes.				
Pass 1. Process input and:				
Pass 2:				
1. CONNECT BOXES[1]				
(Generated)				
2. CONNECT ONE_BOX[3]				
(Generated)				
Pass 3:				
3. TOO_CLOSE1[20] = MIN_SPACING(BOXES[1], BOXES[1], 10/DET)				
(Rules line 26)				
4. TOO_CLOSE2[20] = MIN_SPACING(BOXES[1], ONE_BOX[3], 7/DET)				
(Rules line 29)				
5. NARROW[20] = MIN_WIDTH(BOXES[1], 10/DET)				
(Rules line 32)				
Done.				

Figure 3: Portion of contents of the EXAMPLE.RLO compiler log file.

Note that each rule has been assigned a number. These numbers are occasionally useful.

Preparing the Binary Layout Data File



We create the layout data file in the ICED™ layout editor. Launch the layout editor to edit the file TRIVIAL.CEL. If you use the Windows version, launch the layout editor with the following command in the console window:

ICWIN TRIVIAL

The shapes in this cell should look similar to Figure 4. The cell contains two shapes on layer BOXES: one rectangle and a larger polygon with a 5 user unit "neck". There is also one rectangle on layer ONE_BOX.

Once in the editor, we create the binary layout data for the DRC using the DRC command without any parameters. Type the following command:

DRC

This will export the entire layout contained in the cell to the file TRIVIAL.POK. Once the command is completed, use the **QUIT** command to terminate the editor.

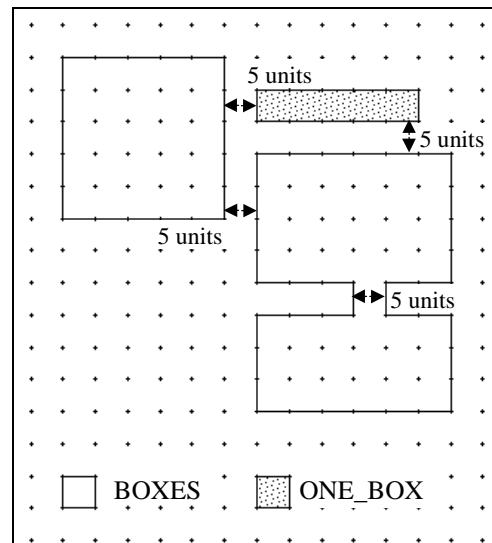


Figure 4: Shapes in TRIVIAL.CEL

Type the DRC command line in a DOS console window or create a batch file.

Now we are ready to run the DRC. The DRC command line is:

DRC3-NT³ EXAMPLE1 TRIVIAL DRCOUT SLOW

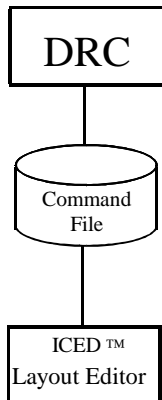
³ The executable file name for released versions for Windows is DRC3-NT.EXE. The executable file name for beta Windows versions is DRC3-NTX.EXE.

The end of the console messages should look similar to:

```
Done.  
100% of chip done.  
***No input skipped.  
***No bad ICED polygons.  
***Error count=7 (raw=10)  
***Total output non-error output count=0  
0 total figures output to non-error layers.  
7 total figures output to error layers.
```

This indicates that the DRC has generated 7 error marks. Most errors generate a pair of error marks. When error marks on the same layer overlap, they may be merged into single shapes. After looking at the error marks, we will see that these 7 error marks represent 4 errors in the layout. We will now cover how to locate these errors using the command file generated by the DRC.

Looking at the Output



To view the errors found by the DRC, we will use the ICED™ layout editor. Launch the layout editor again to edit the TRIVIAL cell. You can use the same DOS command we mentioned earlier:

ICWIN TRIVIAL

Once in the editor, type the following command:

@DRCOUT

This command will execute the DRCOUT.CMD command file generated by the DRC. The error wires shown in Figure 5 are now added to the cell. Your display will probably look somewhat different. We have changed the layer patterns with the LAYER command in the editor to highlight the error marks.

You can change the layer color or pattern of the error wires with the LAYER command if you desire. (See the layout editor reference manual.)

Now we want to select the error wires in the upper right to determine why this error has been marked. Type the following command:

SELECT LAYER 20 IN

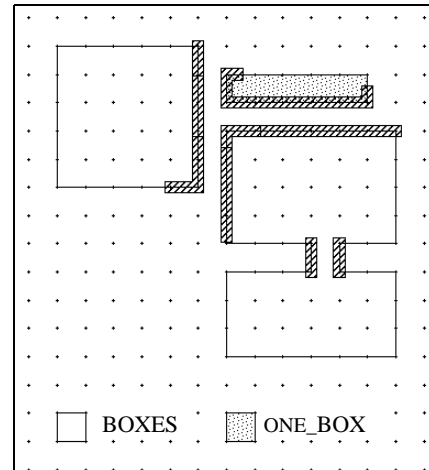


Figure 5: Error marks generated by rules in EXAMPLE1.RUL.

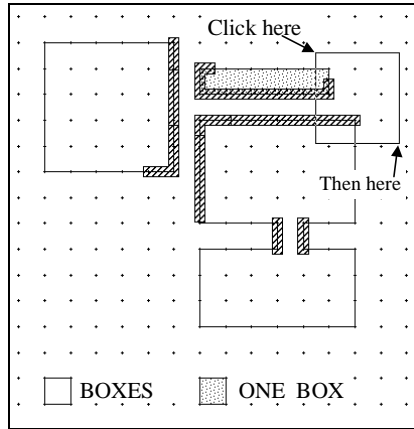


Figure 6: Selecting error wires.

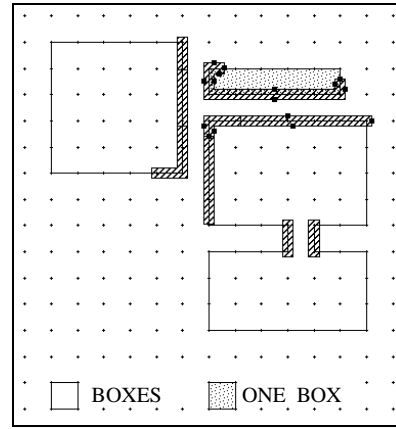


Figure 7: Selected error wires.

Note that the cursor has changed to the select cursor shaped like an 'X'. Click the two points as shown in Figure 6 to select the error wires.

The two error wires should be selected as shown in Figure 7. To see why these error wires were created, type the following command on the command line:

SHOW *

This will report the following information about the selected shapes:

```
ADD WIRE LAYER=20 ID=19 TAG=4 TYPE=2 WIDTH=2.000 AT (10.0,-2.0) &
(10.0, 0.0) (39.899, 0.0)
ADD WIRE LAYER=20 ID=20 TAG=4 TYPE=2 WIDTH=2.000 AT (35.0, 7.0) &
(35.0, 5.0) (10.0, 5.0) (10.0, 10.0) (12.0, 10.0)
```

Figure 8: Information reported by SHOW command.

The tag number refers to the rule that generated the error shapes. If you refer to the rules compiler listing shown in Figure 3 on page 15, you will see that rule number 4 is:

```
4. TOO_CLOSE2[20] = MIN_SPACING(BOXES[1], ONE_BOX[3], 7/DET)
```

This rule indicates that the minimum distance between shapes on layers BOXES and ONE_BOX is 7 user units. The distance between the shapes in the cell is 5 user units.

Note that the DRC has marked an error on the left side of the ONE_BOX rectangle as well. We can assume that this error is caused by a violation of the same rule since this is another case of a shape on layer BOXES being closer than 7 user units to a shape on layer ONE_BOX. If you want to verify this by selecting this error wire as well and repeating the SHOW command, do so now.

We should fix both errors by moving the shape on layer ONE_BOX 2 user units up and to the right. We want to unselect the currently selected shapes, then temporarily hide the error wires to allow us to see the design more clearly by typing the following commands:

```
UNSELECT ALL  
BLANK LAYER 20
```

We need to select the shape on layer ONE_BOX. Try the following command:

```
SELECT LAYER ONE_BOX IN
```

The layout editor will report the following error:

```
SELECT LAYER <<ONE_BOX>> IN  
ERROR: Layer name not in use
```

This is due to the fact that the layer name in the rule set is not the same as that used in the cell. We can see the layer number of layer ONE_BOX in the line from the rules compiler listing shown on page 15. The number in square brackets following the layer name, "ONE_BOX [3]", indicates the number of the layer. **The layer number is always the same in the rule set and the cell**, but the layer name in the rule set can be any convenient string.

In the rule set (shown on page 14) the name ONE_BOX was associated with the layer number 3 in the following rule:

```
Input layer 1 boxes; 3 one_box;
```

To select the shape we need to move, let us select it by number. Type the following command:

SELECT LAYER 3 IN

Now use the cursor to select the rectangle in the upper right of the cell. The shape should be indicated with select marks as shown in Figure 9.

We can move the selected shape 2 user units up and to the right with the following command:

MOVE BY 2,2

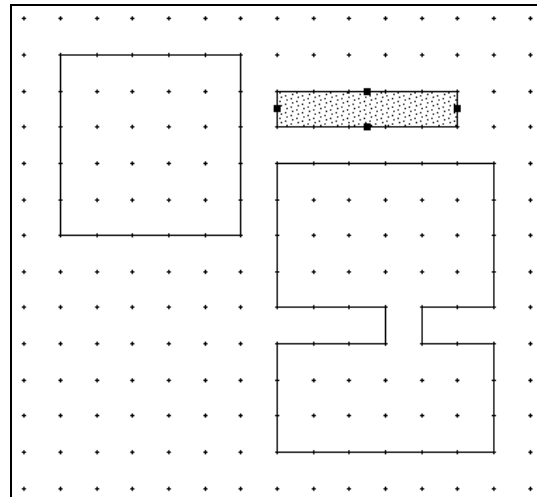


Figure 9: The shape on layer 3 is selected.

This should fix the errors caused by rule number 4.

Now let us unselect the current shape and then see the error wires again by typing the following command:

UNSELECT ALL

UNBLANK ALL

Now let us select the error wires around the neck of the lower shape with the command:

SELECT LAYER 20 IN

Then use the cursor to select the wires in the neck of the lower shape so the two error wires are selected as shown in Figure 10 on the next page.

We can find out the rule number that created these error wires with the command:

SHOW *

You can write the rule set to place error marks from different rules on different error layer numbers. This way you do not have to use the **SHOW** command to see which rule generated each error mark.

When you do this, you will see that the tag number is 5. This means that rule number 5 generated the error wires. From the rules compiler log we see that rule number 5 is:

```
5. NARROW[20] =  
  MIN_WIDTH(BOXES[1], 10/DET)
```

This rule states that shapes on layer number 1 must be at least 10 user units wide. We can see that the neck indicated by the error marks is only 5 user units wide. We must fix this by moving one side of the neck 5 more user units away from the other.

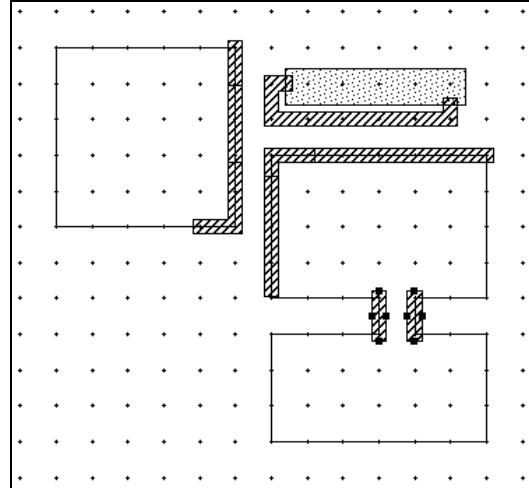


Figure 10: Error wires in "neck" area are selected.

First let us unselect the selected error marks, and then hide all of them with the commands:

**UNSELECT ALL
BLANK LAYER 20**

To select the desired side, use the command:

SELECT SIDE IN

Use the cursor to indicate a very small box around the left side of the neck to select the side shown in Figure 11. Now move this side 5 units to the left with the command:

MOVE SIDE BY -5,0

This fixes the error found by rule number 5.

Let us unselect the side and turn the display of the error marks back on again with the commands:

UNSELECT ALL
UNBLANK ALL

Now let us select the error marks we have not yet looked at with the command:

SELECT LAYER 20 IN

Select the error wires shown in Figure 12.

To determine the rule number that generated this error, type:

SHOW *

The tag number associated with these error marks is number 3. Rule 3 is:

```
3. TOO_CLOSE1[20] =
MIN_SPACING(BOXES[1],
BOXES[1], 10/DET)
```

This rule indicates that shapes on layer 1 must be at least 10 user units from other shapes on the same layer. We can see from the cell that the indicated sides are only 5 user units apart.

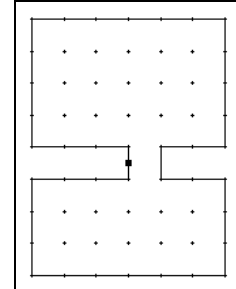


Figure 11

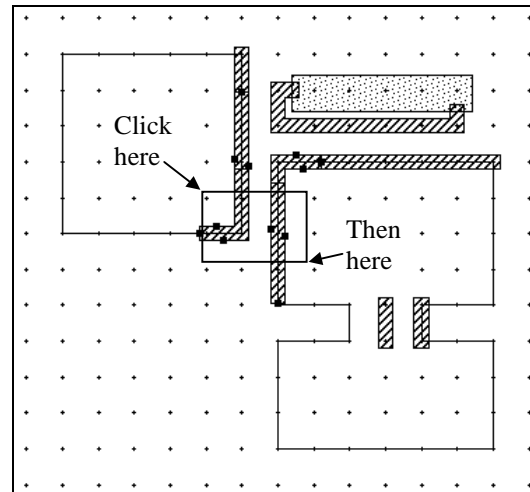


Figure 12: Remaining error wires selected.

Unselect the selected error wires, and hide all of the error wires with the commands:

UNSELECT ALL
BLANK LAYER 20

Now select the side of the lower shape with the command:

SELECT SIDE IN

Use the cursor to select the side indicated in Figure 13. Shift this side away from the other shape on layer 1 with the command:

MOVE SIDE BY 5,0

We have accounted for all error found by the DRC. Let us unblank and delete the error marks with the following commands:

UNSELECT ALL
UNBLANK ALL
SEL LAY 20 ALL
DELETE

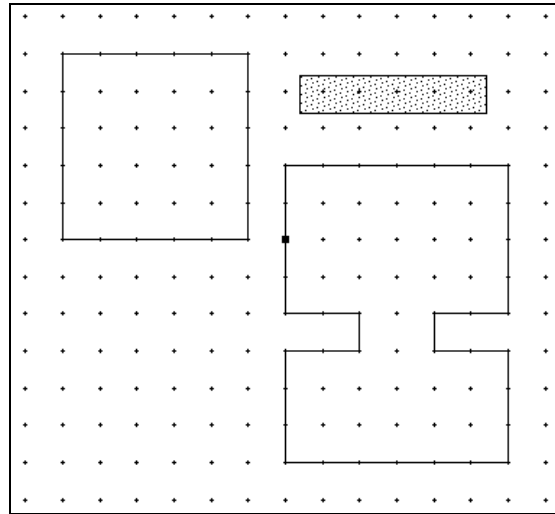


Figure 13: Side of shape on layer 1 is selected.

To test that all errors have been fixed, we will execute the DRC again. It is very important to remember to **always** recreate layout data file (TRIVIAL.POK in this case) when you have changed the design. **If you forget to recreate this file with the DRC command in the layout editor, the DRC will use the same obsolete layout data that you used in the last run.**

Type the command to regenerate the layout data for the DRC:

DRC

We are done with the layout editor. The following layout editor command will save our changes to TRIVIAL.CEL and terminate the editor:

EXIT

You should now be back at the console prompt. Execute the DRC again by typing the following command at the console prompt or in a batch file:

DRC3-NT EXAMPLE1 TRIVIAL DRCOUT SLOW

The DRC program can have several different executable file names. See page 16.

We can tell that no errors have been found by this run of the DRC by the last line of the console messages:

0 total figures output to error layers.

Close the console window by typing EXIT at the prompt or click the 'X' button.

This concludes the tutorial. There are several other sample rule and cell files included in the **Q:\ICED\⁴SAMPLES\DRC** directory. See the table on the next page for a list of these samples and the pages in this manual where they are discussed.

⁴ Remember that Q:\ICED represents the drive letter and path where you have installed the DRC.

Rules file	Related cell file	Rule covered	Page
ALLALL.RUL	TOUCH.CEL	MIN_SPACING	
ALLIN.RUL	TOUCH.CEL	MIN_SPACING /IN	
ALLOUT.RUL	TOUCH.CEL	MIN_SPACING /OUT	
CONTACT.RUL	CONTACTS.CEL	MIN_SPACING /IN	
ENCLOSUR.RUL	ENCL.CEL	MIN_SPACING /IN /OUT	91
EXAMPLE1.RUL	TRIVIAL.CEL	MIN_SPACING and MIN_WIDTH	14
EXAMPLE2.RUL		Removal of redundant rule by compiler	
IN.RUL	TOUCH.CEL	MIN_SPACING /IN	
INALL .RUL	TOUCH.CEL	MIN_SPACING /IN	
ININ.RUL	TOUCH.CEL	MIN_SPACING /IN	
ININI.RUL	TOUCH.CEL	MIN_SPACING /IN	
INOUT.RUL	TOUCH.CEL	MIN_SPACING /IN /OUT	
INOUTI.RUL	TOUCH.CEL	MIN_SPACING /IN /OUT	
LEXAMPL.RUL	LEXAMPL.CEL	MIN_SPACING /LENGTH	
NOTCHSP.RUL	WIRECELL.CEL	MIN_SPACING (does not mark spacing problems in single shape) MIN_NOTCH	87
OUTALL.RUL	TOUCH.CEL	MIN_SPACING /OUT	
OUTIN.RUL	TOUCH.CEL	MIN_SPACING /IN /OUT	
OUTINI.RUL	TOUCH.CEL	MIN_SPACING /IN /OUT	
OUTOUT.RUL	TOUCH.CEL	MIN_SPACING /OUT	
OUTOUTI.RUL	TOUCH.CEL	MIN_SPACING /OUT	
OVERLAP.RUL	GATE.CELL	MIN_SPACING /IN /OUT	

Figure 14: Rules files included with DRC installation

Troubleshooting

If the DRC crashes while you are trying to run the tutorial test case, or at any later date, try to look your problem up in the table below. **Try the recommended fixes before you call technical support.**

Symptom	Possible Cause	Recommended Fix	Refer to page
DOS version crashes on Windows operating system	Long directory or file names	Best Solution: Change to Windows version. Otherwise, rename or move directories (including installation directories and data directories) to names that have no more than 8 characters in any one string. Do not use blanks in names. Use file names that follow the classic 8.3 format (no more than 8 characters before the '.' and 3 characters for the extension). Do not use blanks in names. Cell names do not have to be less than 8 characters, only file names.	N/A
	Long environment variables	Best Solution: Change to Windows version. Otherwise, ensure that all environment variables have no more than 127 characters in the entire <i>var=value</i> string. Use DOS SET command to verify and/or change values.	359
	??	Change to Windows version. Sometimes the DOS version will not run under Windows for other reasons that cannot be fixed. If the two other fixes above do not fix the problem, you must change to the Windows version.	N/A

Symptom	Possible Cause	Recommended Fix	Refer to page
Mysterious crash very early in run	Not enough memory to start program	Check top of log file for actual amount of memory available to the DRC. Try USE or HOG options to reserve more memory for program. Reboot or free more memory in operating system.	339
Options with '=' in the command line not read correctly	Operating system command line parsing	Replace '=' with '#' in options where the '=' is removed by operating system before the DRC is passed the command line. The DRC considers the two equivalent, but the '#' symbol is not stripped by the operating system.	333
Copy protection (DRCnAUX.EXE) fails in Windows	Operating system incompatibilities	Obtain new copy protection package from IC Editors, Inc.	10
MIN_SPACING not marking errors between shapes on same layer.	Polygons are merged to form single shape.	Add a MIN_NOTCH rule to test spacing between fingers of a single shape. All touching shapes on a single layer are merged and MIN_SPACING marks only errors between different shapes.	87
MIN_SPACING not marking other errors.	QUICK_SPACING option hiding errors	Remove QUICK_SPACING from command line.	100
Hierarchical output is not generating shapes in the correct cells.	Cell flattening on input, safe processing, etc.	Quirks of hierarchical processing are covered on the indicated page.	149

Symptom	Possible Cause	Recommended Fix	Refer to page
Some rules are not being executed	QUICK-PASS option	Remove from command line. This option speeds up DRC runs at the cost of not executing rules that require more than one pass.	337
	DO option	Remove from command line. This option restricts which rules are executed at run time.	347
False errors	Resolution grid issues	Vertices of output shapes may be shifted to lie on more restrictive resolution grid.	79
	Panel processing	Shapes that are cut at panel boundaries may result in acute angles. QUICK_PASS option may be forcing problems at panel boundaries.	76 129
	Other causes	See guide to removing miscellaneous false errors.	156

Fundamentals of Design Rule Verification

What Are Design Rules?

Each integrated circuit technology has design rules determined by the fabrication process used to manufacture the chip. Factors like the resolution of the photographic process and misalignment of layers during manufacture require that shapes on certain layers have a minimum width, are a certain distance apart, etc.

The engineers who develop new technologies usually specify these design rules. These specifications are documented and distributed to chip developers. Designs that have violations of these design rules will often fail to perform as expected.

Some chip designers modify these design rules for certain purposes:

More stringent rules may be required to insure a longer life for the chip, or because of the uniqueness of a particular design.

If you are developing a standard cell library, you may need to add restrictions so that you can verify that all cells obey the added requirements required for features like interchangeability or automatic placement.

Some designers may need to relax certain criteria in some areas of the design to get the very best performance or density from the technology. These types of modifications to the design rules should be made only after careful consideration and review.

How Design Rules Are Verified

Minimum Spacing Rules

In simple terms, a minimum spacing rule is a rule that checks that shapes on one layer are far enough apart from other shapes on the same layer, or from shapes on a second layer. Due to the ever-increasing density of modern integrated circuits, spacing criteria are increasingly complex.

What design rule verification programs actually check is the distance from a given side of a shape to sides of shapes on the other specified layer. Therefore, one pair of shapes may result in several violations of a spacing rule if several different pairs of sides are too close.

Simple spacing rules that verify that each side of one shape is at least a minimum distance in all directions from a second, non-overlapping shape are easy to understand. However, to allow as much density as possible, minimum spacing rules are often modified to allow exceptions in certain situations.

Inside/outside criteria: These types of exceptions restrict the spacing rule to sides of shapes that are found looking toward the interior of a shape, or toward the outside.

End caps: Often the minimum spacing is critical only along the length of a side. Sides a minimum distance away from an end vertex in a diagonal direction are acceptable. This area around the end vertex of a side is called the end cap of the side.

Intersections of sides: Sometimes the direction of sides is critical to a design rule. Wires that cross may be acceptable, while wires that travel parallel to each other need to be a minimum distance away.

Electrical connections: Whether or not two shapes are part of the same electrical net is often important to spacing rules.

Minimum length: Many design rules allow very short violations to be present, however, longer violations must be fixed.

Other Verification Rules

Most other design rules are more straightforward than spacing rules. A common category of rules insures that shapes on a given layer are not smaller than a certain minimum dimension. However, even this simple concept can require some careful thought to write design rules that will catch all possible problems.

A common rule in this category is a minimum width rule. This type of rule insures that a polygon is at least a minimum distance wide in any direction. The primary reason for this type of rule is that opens may occur due to processing issues if a shape is too small at any point.

Verifying this type of rule comes down to testing each side of a shape to see if it is too close to another side. However, this is more complicated than it seems at first.

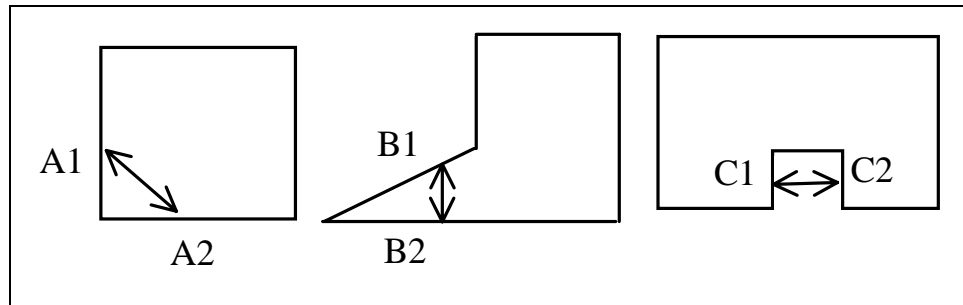


Figure 15: Various side-side distances.

Refer to the figure above. Sides A1 and A2, or any adjacent pair of sides, will violate a minimum width rule if the program does not prevent adjacent sides from being marked as errors. You can see that adjacent sides cannot be considered a violation. However, in this case, sides B1 and B2 cannot be considered to be a violation. When sides are allowed to be at any angle, this type of problem must be found with a different rule.

You can see that you must understand how a program verifies even such a simple rule as a minimum width check to insure that all violations are found.

Sides C1 and C2 may be closer than a minimum distance, however they do not represent a minimum width violation. To avoid marking this pair of sides, a program must consider whether the distance is measured through the inside of the polygon, or across a gap.

The gap between sides C1 and C2 is called a "notch". Minimum notch rules are also common in some technologies. Minimum notch rules have all the same problems in verification that minimum width rules have.

It is often important to test more than just minimum width for some layers. Most devices must be tested to insure that they have at least a minimum area in addition to being a minimum width wide. Occasionally, it is important to test each side to insure that it is at least a minimum distance in length. This type of rule is usually referred to as a minimum side rule.

The software used to create masks often adds additional design rules. Configurations such as self-intersecting shapes and acute angles must often be eliminated before a layout can be used to create a mask set. The hole that is left when a shape intersects itself can often not be fabricated properly. Acute angles can result in unexpected results when a shape is bloated prior to creating a mask. Bloats of acute angles will make the shape grow disproportionately long.

Eliminating False Errors

A false error is a shape or relationship between sides that violates the design rules, but for some reason is not considered a real error.

One cause of false errors may be that the design is incomplete. Let us say that you are verifying a subcell. This cell will be connected to other cells with wires at a higher level in the design. You have rule that states that all contact shapes must be covered by metal, but the metal is not contained in the subcell. It will be added in the higher level cell.

When you run the rules on the subcell, violations will be marked for all uncovered contact holes. You consider these to be false errors. One way around this problem is to simply ignore the false errors. However, if the subcell is a

large complicated cell, and you have many false errors that you ignore, it is very easy to ignore some real errors as well.

One way to avoid false errors is to add shapes on non-design layers (often called dummy layers) to the design. You could add shapes on layer TEMP_METAL that cover the contact holes, then modify the rules to say that contacts covered by TEMP_METAL are not errors.

However this type of processing has risks as well. Let us say that you forget to remove the shapes on TEMP_METAL when you are testing rules on the higher level design. You have real uncovered contacts at this level, however the errors will never be reported since you are still using the TEMP_METAL processing.

When you add shapes on dummy layers, you should always think carefully about how they may prevent real errors from being found. Remove shapes on dummy layers, or add methods to insure that they are not hiding real errors, before you run design rule verification on the final design.

A more risk-free method to prevent the false errors mentioned above is to create a temporary cell with shapes on the real metal layer, then add your subcell to this cell. Output the data to test the subcell from this temporary cell. This method requires no non-design shapes in your design cell, and no risk of missing real errors at the higher level.

Other false errors that do not result from missing shapes at a lower level may require other solutions. Let us say that wide metal wires must be at least 5 microns apart, but narrow wires can be 3 microns apart. If your rules test that all metal wires are at least 5 microns apart, you would consider error marks on some narrow wires to be false errors. However, if you just try to ignore these errors, it is very likely that you will miss real spacing errors hidden among the rest.

To solve this problem, you need to filter the metal layer into groups that are tested with different rules. When the rule set takes into account why some shapes should be tested with different rules, the rule set is more risk-free. In the case of the metal wires, you can use layer processing to classify the wires and put wide wires into one layer and narrow wires into another. Then write separate spacing rules for each layer. (An example of this process is covered on page 65.)

Layer Manipulation Prior to Rule Verification

It is almost always required to process the layers in a design before they can be verified. Even if false errors are not an issue, you must often combine or filter layers to isolate shapes that represent devices or other special shapes that must be checked with a different set of design rules.

Sometimes the layers you use in the layout must be bloated, shrunk, or combined with Boolean operations to simulate the fabrication process before the design rules can be applied.

For example, if you are building a design in a FET technology, you must create the gate layer using a Boolean AND operation on the diffusion and polysilicon layers. The gate layer device shapes are very likely to have different design rules than either the diffusion or polysilicon layers.

Some design rules are tested with Boolean operations or with rules that test if one shape touches another. This can be an important distinction. Let us use the example of contact holes again. Consider a restriction that all contact holes must not only be covered by metal, but also must be surrounded by a non-zero amount of metal on all sides. This is sometimes referred to as **enclosure verification**.

Look at Figure 16. Case 1 is the only case where the contact hole does not violate the rule stated above. If you use a Boolean rule to test this restriction, such as "CONTACTS AND NOT METAL", it will find cases

2 and 3, however case 4 will not be found. Case 4 can be tricky to find even with a spacing rule. There are cases where complex spacing rules will not mark coincident edges. It is best to find this type of error with a pair of rules such as the following:

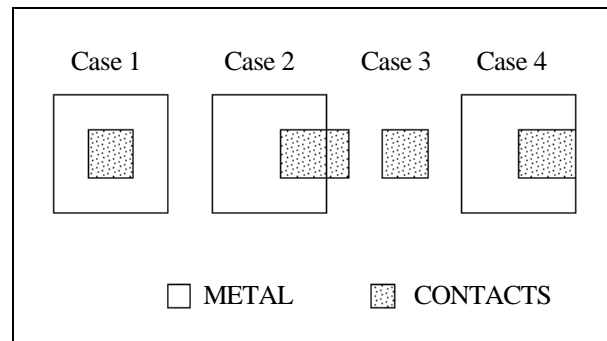


Figure 16: Contact hole positions.

NOT_METAL = NOT METAL
ERR = CONTACTS TOUCHING NOT_METAL

Creation of Layers for Import Back Into Design

The same layer generation methods that combine or transform layers before design rules are verified can be used to manipulate layers for import back into design cells. This allows you to perform layer manipulation that cannot be accomplished with the functions available in the layout editor.

You can use this feature to generate mask layers from simpler layers in your design. Layers like diffusion layers or wells can be far easier to design if you use layer manipulation in the verification program to generate the layers exactly to specification. The layout designer does not have to worry as much about following the design rules for layers like this, since they will be generated automatically based on the presence of other layers.

Generated layers can be created and then verified with design rules in the same rule set. The rule set can be written to not only verify the new layer, but to automatically modify it to obey the design rules.

How Do Design Rule Checkers Work?

The basic flow of data into and out of a design rule checker is the same for all programs of this type. There are two primary input files: the design rules and the layout data.

The design rules are translated from written specifications to the programming language of the design checker. This step can be complicated and requires quite a bit of careful thought. The most dangerous mistake beginners make is assuming that stupid mistakes will not be made in the layout. You must assume that every possible error will be present in the layout, no matter how silly it may seem. Layout designers often turn the display of most layers off as they design a layer, so even the most obvious problems between two layers are not visible to them as they are working. Problems like shifting a shape or a cell often cause problems beyond the portion of the design visible on the screen. Everybody makes mistakes, and the writer of the design rules must insure that **all** of them will be found.

See the NO_RUL rule description for details on how the DRC verifies that a modified rule set has been recompiled.

Some programs (like the DRC) compile the source rules set into a compact form that allows the program to execute faster. Since this an extra step, you must be careful to remember to compile a modified rule set before executing the design checker again. By default, the DRC will warn you if your rule set has changed since the last compile.

The layout data is usually exported from the layout editor into a compact machine-readable format. This is also an extra step that you must be careful not to forget, otherwise after you make a change to the layout, you will verify the old data again rather than the updated layout. There is no way the DRC can warn you if one of your cells has changed since the last time you created the DRC data.

The primary output from a design rule checker is a report of all violations of the design rules found by the program. It is easiest to locate these errors if this report is in the form of graphic data that enables you to see the errors in the layout editor rather than finding them from lists of coordinate data. This is the method used by the DRC.

A Few Definitions

Before we go into the details of how design rule checkers process data, let us define a few terms.

Cell A cell is a collection of shapes that is stored as a group with a name associated with it.

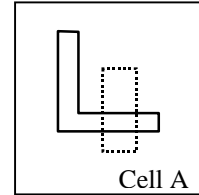


Figure 17: A cell.

Hierarchy This term refers to the nesting of cells in a design. When cell A is added as a component to cell B, we say that cell A is **nested** in cell B, or that cell A is a **subcell** of cell B. Cell B has hierarchy, or is hierarchically nested. That is, it contains other cells as components.

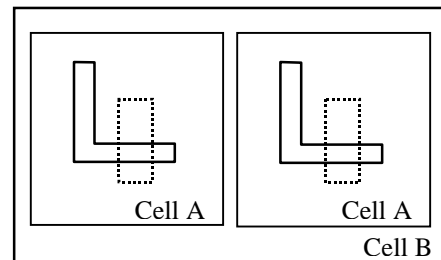


Figure 18: A nested cell.

Level The level of a cell refers to its place in the hierarchy. In the example above, if cell A has no cells nested in it, it is the **lowest level cell**. Cell B is a higher level cell. If cell B is added to cell C, and cell C is not added to any other cells, it is the highest level cell, or the **main cell**.

Flattening When you replace a nested cell with the shapes contained in the cell, you have flattened it. **Ungrouping** is another term for flattening.

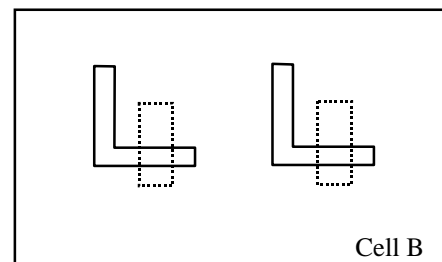


Figure 19: Cell B after flattening.

Pass A pass is a collection of operations that can be performed by a design rule checker with one sweep through all shapes in the database.

Sometimes, the rules require that all shapes must be processed several times to complete all operations. We will discuss later what types of operations require this type of processing. For now let us just say that if some types of operations were executed with a single pass through the data, errors could be missed, or false errors could be generated.

How Large Amounts of Data Are Processed

One of the biggest problems with design rule checkers is the huge amount of data involved in verifying the design of an entire chip.

To reduce this problem, and speed the verification of your chip, it is best to verify smaller portions of your design first. The time and storage required to verify the design rules for a NAND circuit is negligible. If you find most of the design errors in subcells first, there will be fewer runs required on the entire design. However, you **must** run the design rules checker on the entire chip before you can release it for fabrication. Areas where cells meet or overlap are common places for design rule violations.

Other programs that verify design rules allow true hierarchical verification by imposing design constraints. These constraints are usually forbid cells to overlap. Wiring in the main cell is not allowed to travel over subcells. Some programs allow some overlap by forcing all devices to be in one area of a cell with only wiring and contacts allowed in a border area. This border area may overlap wiring in the main cell. Both of these methods waste design space.

The DRC program imposes no design constraints. Any customization allowed by the technology is supported. When no constraints are imposed, a design rule checker must flatten a design prior to verifying the design rules to find all errors where cells meet or overlap.

For example, let us say that your design contains 10,000 NAND cells. These cells are added to a cell called WIRING that contains metal wires that connect the NAND cells. It is not sufficient for a program to verify a single copy of the NAND cell, then verify the wire shapes that are contained in the WIRING cell. There may be design rule violations between shapes in the NAND cells and shapes in the WIRING cell. Each copy of the NAND cell must be flattened (or unnested) so that every shape in each copy of the NAND cell is verified against the wire shapes in its vicinity.

When you flatten a hierarchically nested design, the amount of data increases dramatically. In the example above, if we assume that the NAND cell contains 50 shapes, and the WIRING cell contains 20,000 shapes, the flattened design contains 520,000 shapes. Just one layer of a modern dense chip design may contain millions of vertices.

In addition to the coordinates of every vertex of every shape in the design, the program stores temporary data for every vertex to enable the program algorithms to execute at reasonable speeds. When an entire chip is verified, the huge amount of data that must be stored is usually far too much for a personal computer.

Panel Processing

The solution to the storage problem for a whole chip is to divide the design into panels and process only a single panel at a time.

The portion of the design that is not included in the current panel can remain hierarchically nested to conserve storage space. In the example above, if each panel contains only 50 copies of the NAND cell, only 50 copies must be flattened at a time, resulting in a database of only 22,500 shapes.

Panel processing has overhead associated with it. An area outside the boundary of each panel must be tested to insure that no violations occur between the shapes in the current panel and its neighbors. This area is called the panel border.

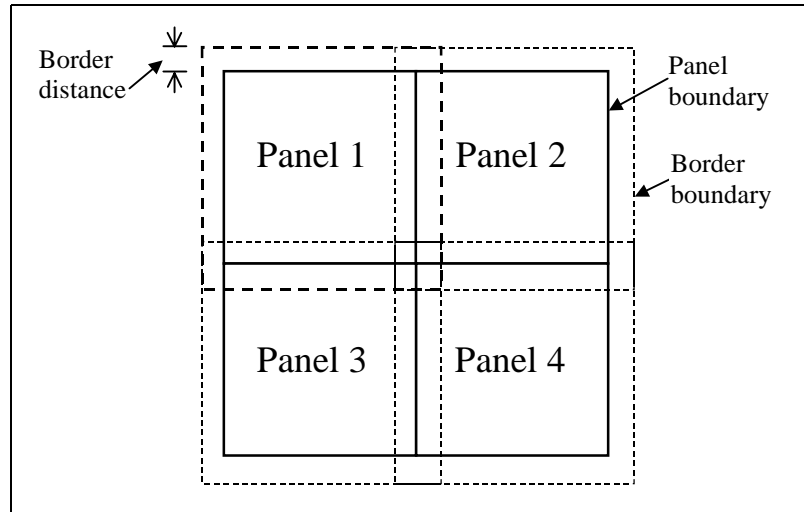


Figure 20: Panels and borders

The shapes in the panel borders are tested at least twice. The shapes near the interior corner of a panel are tested at least 4 times. In the diagram above, a shape in the lower right corner of panel 1 will be verified once when panel 1 is checked, then again when panels 2, 3, and 4 are checked.

The border distance is determined by the design rules. If your rule set contains a single rule that verifies that each shape on the metal layer is at least 2 units away from all other shapes on the metal layer, the border distance must be at least 2 units. In this case, if a shape on the metal layer is right at the panel boundary, a violation in a neighboring panel that is less than 2 units away will be found.

There is a tradeoff between the storage savings in testing one panel at a time and the overhead in testing the border area multiple times. The optimum panel size for a particular design and rule set can be determined only by trial and error.

Hierarchical Processing

A design must be flattened to verify design rules. However, many layer-processing operations that must be performed prior to applying the design rules can be performed while the design is still nested hierarchically.

Remember the chip with 10,000 NAND cells? Let us say that this NAND cell needs to have the GATE layer generated from the diffusion and polysilicon layers to test FET device design rules. The generation of the GATE layer can be performed only once on the nested NAND cell, rather than 10,000 times. Only one copy of the shapes on the GATE layer must be stored. This will not prevent design rule errors from being found.

Each pass through the data requires that intermediate data must be stored for the next pass. If the design is being processed in panels, the program cannot test the GATE layer in the same pass that generates it. All data for the new layer must be stored until the next pass can use it. When you have many generated layers, and several passes, this amount of data can be huge. If it is stored hierarchically, significantly less storage space is required.

There is overhead incurred when processing layers in this manner. The layer stored for a higher level cell must have the data in the lower level cell subtracted from it, or you may generate several copies of the shapes. These subtractions take processing time. Also, shapes in a higher level cell may modify a layer stored in a lower level cell. This must be checked at every level, or incorrect results may be generated.

Due to these types of problems, there is trade off between time saved by hierarchical processing and the time spent in the extra calculations. The best solution is to flatten small cells and those cells used few times.

Now that we understand most of the issues involved in testing design rules, let us go into the specifics of how the DRC deals with these issues.

How the DRC Works

How the DRC Works

This chapter is an overview of how the DRC operates. Once you have completed this material, you should have at least a basic understanding of all features.

The theory behind the more complex features is described completely. To use these features, you will have to read the relevant syntax sections. However, this chapter covers the key ideas that will allow you to solve problems you may encounter, and to avoid all major pitfalls.

Every subject is followed by a table of references to other places in the manual where you can find more details and examples. You may find these tables especially useful if you need to review a particular subject in detail at a later date.

Generating the Input Files and Running the Program

The DRC requires two input files: a compiled rules file and a binary layout data file.

The source rules file can be written using any ASCII text editor. We suggest that you use a .RUL extension for these files.

The rules file must be compiled with D3RUL-NT.EXE⁵. The compiled rules file will be created with the same file name as the source rules file with a file extension of .BB.

A few hints on using the DRC layout editor command are provided on page 159.

The binary layout data is created by the DRC command in the ICED32™ layout editor. A complete description of the DRC command is provided in the layout editor reference manual.

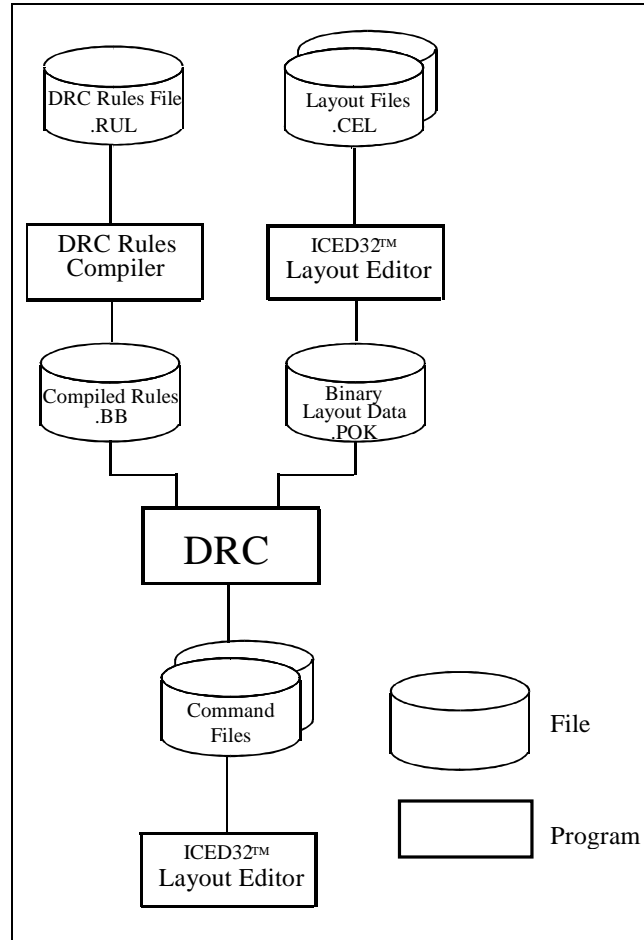


Figure 21: The flow of data to run the DRC.

⁵ The executable file for released versions for Windows is D3RUL-NT.EXE. The executable file for Beta Windows versions is named D3RU-NTX.EXE.

How the DRC Works: Generating the Input Files and Running the Program

Once both of the input files are prepared, the DRC program is executed outside of the layout editor. There are many command line options for the program. You should be familiar with them before executing the program. The purpose of each option is mentioned below in this chapter, but they are more fully described later in the manual when we discuss the syntax of the DRC command line.

The DRC program generates few console messages. The final line of a successful run is the most important one.

n total figures output to error layers.

If $n = 0$, then the DRC has found no errors in your layout. (We will describe exactly what an error layer is in a few pages.)

Subject	Description	Page
Rule syntax	Detailed syntax and examples for every rule in alphabetical order	171
NO_RUL rule and NO_RUL command line option	Avoid warning prompt from DRC when source rules file is not present	277 and 349
Rule compilation	Details on compiler and syntax of command line	319
DRC program	Details on DRC command line syntax and output files	329

Figure 22: References for running the program

Looking at the Results

The third parameter on the DRC command line (*output_file_base_name*) is the string used to create the two primary DRC output files: the log file and the command file.

The DRC Log File

You will be warned in the console messages when you need to check the log file for warnings or error messages from the program.

The log file will have the name *output_file_base_name*.DLO. The log file contains all messages about program errors or warnings. Some other information provided in this file includes details on how the design was flattened, the panel size, and the border dimensions.

There are some DRC command line options that will add information to this file. See the list below.

The number of shapes created on each output layer, including error layers, will be listed in the log file. This is a quick way of seeing how many violations were found by each design rule checked. **The coordinates of each violation are usually not listed in the log file.** Since the DRC is designed to verify large designs, if all violation coordinates were printed, the log file could grow to be unreasonably large.

Subject	Description	Page
Log file description	Detailed description of contents	362
LIST_RULES	DRC command line option to add rule listing to log file	350
SHOW_BORDER	DRC option to add border calculations to log file	348
SHOW_SCALES	DRC command line option to add vertex resolution parameter information to log file	350
DRC program	Details on DRC command line options and output files	329

Figure 23: References for DRC log file

Detailed Logging

You can enable detailed logging of violations for some rules. Detailed logging may be helpful when you cannot determine why a particular side has been marked as an error. The coordinates of pairs of sides that violate the rules listed in Figure 24 will be listed in the log when detailed logging is enabled.

MIN_NOTCH MIN_SPACING MIN_WIDTH

Figure 24: Rules that can produce detailed listings in log file.

Use detailed logging **only for small areas** of your design when you cannot determine the exact cause of an error. The log file gets unreasonably large quickly when this feature is used.

The more common way to mark errors is the command file that creates shapes in the layout editor. (We cover this subject next.) If many error marks are generated, it can be difficult to determine from these error marks which specific pair of sides violates a rule. Since the detailed log messages explicitly list each pair, this can help you pinpoint a problem.

The marks in the command file mark only the portions of sides that violate the rule. However, the messages printed when detailed logging is enabled are the coordinates of the entire side, not just the portions in error.

<pre>4. RESULT1[11] = MIN_SPACING(A[1], A[1], 20 /+~CONN/P/INTER/OVER/CROSS/T/END/DET) 1: 1 (-29.5,19.5)-(-29.5,-5) <-> 2 (-30,-10)-(-20,-10) 2: 1 (-29.5,19.5)-(-29.5,-5) <-> 2 (-30,-10)-(-30,-15)</pre>

Figure 25: Sample log messages produced when detailed logging is enabled.

All violations for a specific rule will be preceded by a listing of the rule. The rule number (“4” in the sample above) is indicated first.

Each violation message begins with an incremented error number. Next a unique number for the polygon containing an edge in error will be listed. The vertex

coordinates for the edge in error is printed followed by the symbol “<->”. Next comes the number of the polygon containing the edge that was too close, and finally the coordinates of the vertices of this edge. Figure 25 above shows two error messages printed for violations of a MIN_SPACING rule. A single edge of polygon number 1 is too close to two different edges of polygon number 2.

There are two ways to enable detailed logging. The rule DETAIL ON enables detailed logging until a DETAIL OFF rule disables the feature. The other method is to add the /DET option to specific rules. (The rules listed above in Figure 24 all have this option.) If detailed logging has been enabled with DETAIL ON, the /~DET option in a rule disables detailed logging for only that rule.

Subject	Description	Page
DETAIL ON/OFF	Rule to enable/disable detailed logging	210
MIN_NOTCH	Spacing rule for notches	248
MIN_SPACING	Spacing rule for separate shapes	252
MIN_WIDTH	Spacing rule for width of individual shapes	271
Limiting Area Checked	Overview of methods to check only part of a layout.	159
TOP, BOT, etc command line options	Restrict area to check on the command line at run time.	350
IN option of layout editor's DRC command	Restrict area to check in input layout file.	159

Figure 26: References for detailed logging

The DRC Command File

The coordinates for each error shape are included in the commands in the command file. You can browse this file to read the coordinates if you desire.

The command file is used to create shapes in the ICED™ layout editor. The DRC will create ADD commands in the command file for all shapes on output layers at the end of the DRC run. These shapes will include both error marks and shapes on all other output layers. You execute this command file in the layout editor to add the shapes to a cell.

The name of the file is *output_file_base_name.CMD*, where *output_file_base_name* is the third parameter on the DRC command line. You execute the command file in the layout editor with the command:

@output_file_base_name

The shapes will be created in the current cell. (Except in the case of hierarchical output. We will cover this special case later.) If the current cell is your design cell, it will be modified when you execute the above command. We suggest that you execute the command file in a temporary cell until you are familiar with the process. You can add your design cell to this temporary cell to see the error marks at the same time as your design.

See page 62 for a chart that indicates which rules generate wires and which generate polygons.

Most verification rules generate wires for error marks. This is due to the fact that wires can clearly mark the sides of polygons. Only the portions of sides that are in violation of a rule will be marked. If the DRC marked entire polygons, it would make the error marks much more ambiguous.

See an example of setting wire width and wire type on page 367.

One helpful feature of wires is that you can set their width to be consistent with the size of the shapes they are marking. Unless you use the WIRE_WIDTH rule or the WIRE_WIDTH command line option, the command file does not set the size of the wires it creates. The default width for the layer in the cell file will be used as the width of the wires. If you have shapes that are usually 10 user units wide, you can set the width of the layer for the error wires to 1 or 2 user units to clearly mark the edges. If you are marking errors on shapes that are only 2 user units wide, you can set the default width of the error layer to .3 user units.

You can set these layer properties in a command file that you can reuse as required. See page 355 for details on executing such a command file automatically when you execute the DRC command file.

It is difficult to change the width of wires after they are created. You want to set the default width of your error layers **before** you execute the command file. The LAYER command in the layout editor sets the default width. You also set the color and fill pattern of the layer with this command. The default end type of all wires is set with the USE command. You should always set the default end type to type 0 wires before executing the command file.

If you do not want to customize the width for every error layer in the cell file before you execute the DRC command file, you should add a WIRE_WIDTH rule to the rule set to set the width of all error wire marks.

Some errors are marked in separate subcell error command files. The errors for a specific subcell will be created in a separate file with a .ERR file extension. The shapes are created in the coordinate system of the subcell. These files facilitate fixing the type of errors found in nested cells. (Remember that most errors can be found only after the layout has been flattened.) Execute these command files while editing the corresponding subcell.

Subject	Description	Page
Command file description	Detailed description of contents and how to import shapes into the layout editor	365
WIRE_WIDTH rule and command line option	Set width for all error wires created by the command file.	315 346
Simple tutorial	Example of use of command file	12
Hierarchical output	Overview of how hierarchical output is handled in command file	146
Subcell error command files	Detailed description of .ERR command files	375

Figure 27: References for DRC command files

Additional Uses of the DRC

In addition to finding design rule violations, the DRC can be used to manipulate layout data. You can use the layer manipulation operations to transform input layout data into mask layers or other useful layers.

See an example of how useful this feature can be on page 71.

Combining the DRC with the ICED™ layout editor allows you to perform operations too complex for the layout editor alone. These operations include:

- Boolean operations

- Shrinks or bloats

- Isolation of subsets of shapes by size, touching criteria, or other characteristics

The output data created by these operations can be hierarchical. This means that the structure of a nested design can be preserved in the output data.

One more feature of the DRC is the ability to compare two designs. The `SECOND_CELL` command line option allows you to compare two layouts or to combine the data in two cells with the control available with the layer generation operations mentioned above.

Subject	Description	Page
Layer manipulation	Overview of rules that manipulate layers	63
Hierarchical output	Overview of how hierarchical output is handled	146
Mask layer generation	Overview of mask layer generation issues	70
SECOND_CELL	DRC command line option to import second layout file	335

Figure 28: References for additional uses of the DRC

Layer Processing

Layer Definition

Use the LAYER or TEMPLATE commands in the ICED™ layout editor to determine the layer number from the layout cell's layer name.

All layers in a DRC rule set must be defined before they are used in a rule. These definition rules associate the DRC layer name in the rule set with the layer number in the ICED™ cell. The only place in the rule set where the layer number is used is the layer definition rule. The rest of the rule set uses only the DRC layer name. **The name of the layer in the ICED™ cell is ignored completely by the DRC.**

DRC layer names may be up to 30 characters long. The first character must be a letter. The remaining characters can be letters, digits, or any of the following special characters: '_', '~', '\$', '!', or '#'.

You may define up to 2100 unique layers in a rule set. Layer definitions are usually grouped together at the top of the file, but this is not required.

Name of rule	Use	Page
INPUT LAYER	Defines input layers	217
OUTPUT LAYER	Defines output layers	284
MODIFY LAYER	Defines layers used as both input and output layers	273
SCRATCH LAYER	Defines temporary layers used in the rule set	300

Figure 29: Layer definition rules

Input Layers

Use the assignment rule to copy a layer. See page 187 for details.

Input layers correspond to layers in the input ICED™ layout. Only input layers will be read in from the layout data file. Other layers are ignored. Input layers cannot be modified during the rule set (unless they are defined by a MODIFY LAYER rule. This is covered on the next page.). If you want to modify the shapes on an input layer, you must first copy the input layer to an output or scratch layer.

Output Layers

Shapes on output layers will be included in the command file generated by the DRC. Shapes on other layers in the DRC database will be discarded at the end of the DRC run. Output layers can be either error layers or non-error layers. We will cover this subject a little later.

The OUTPUT LAYER rule assigns a layer number to the DRC layer name used in the rest of the rule set. Multiple DRC layers can all be combined into one output layer number at the end of the DRC run. The layers are processed separately during the run.

Output layers can be useful for diagnosing problems with your rule set as well as finding errors in your layout. If the rule set is not creating the shapes you think it should, or not marking errors you think it should, the problem is often that the rules that process intermediate layers are incorrectly written. When you look at these intermediate layers the problem is often quite obvious.

See an example of this type of scratch layer definition on page 152.

One syntax trick you can use to make your rule set easier to troubleshoot is to define all intermediate or temporary layers in OUTPUT LAYER rules using layer number 0. Shapes on layer number 0 are not included in the output. However, it is easy to edit the rule set later to temporarily set specific layers to a layer number other than 0. When the layer number is non-zero, shapes on the layer will be included in the output.

In other words, layers defined in OUTPUT LAYER rules with layer number 0 are really scratch layers which will not be included in the output. The other method used to define scratch layers is described next.

Scratch and Modify Layers

Any layers that the DRC will create or modify, that are not output layers, must be defined in a SCRATCH LAYER rule. These layers are defined only by name since they are never output to an ICED™ cell as numbered layers. As mentioned in the previous paragraph, you may prefer to use OUTPUT LAYER 0 rules for scratch layers instead of using the SCRATCH LAYER rule.

A layer defined with the MODIFY LAYER rule is both an input layer and an output layer. Modify layers are useful when you are using the DRC to create new cells. You can list all layers in a cell in MODIFY LAYER rules, then all layers will be included in the output data using the same layer numbers.

However, it can be dangerous to use modify layers when you import the DRC output into your original cells. When you do this, shapes on all layers defined with MODIFY LAYER rules will be added to the original cells.

For example, you have a cell with two shapes, one on layer 1 and one on layer 2. In the DRC rule set you define layers 1 and 2 with MODIFY LAYER rules. Your rule set uses these two layers to create layer 3. When you import the DRC results into your original cell to see layer 3, you are also adding copies of the shapes on layer 1 and 2.

Variable Layer Numbers

Occasionally, you may want to specify layer numbers when you run the DRC rather than in the rule set. One case of this is when you have a simple rule set that performs a Boolean operation on two input layers. If you can specify the layer numbers on the DRC command line, you do not have to edit the rule set, then recompile it, to perform the same operation on other layers at a future time.

How the DRC Works: Layer Processing

The %n syntax used for variable layer numbers is designed to be similar to the syntax used for parameters in DOS batch files.

You can use variables in place of layer numbers in INPUT LAYER, OUTPUT LAYER, or MODIFY LAYER rules. Simply use a percent sign (%) and a counter in place of the layer number in these layer definition rules. Then specify the layer numbers for each variable on the DRC command line using the LAYERS option.

Subject	Description	Page
INPUT LAYER	Define input layers	217
OUTPUT LAYER	Define output layers	284
MODIFY LAYER	Define layers used as both input and output layers	273
SCRATCH LAYER	Define temporary layers used in the rule set	300
LAYERS command line option	Define layer numbers used for input/output at run time instead of in rule set	346

Figure 30: References for layer definition

Preprocessing of Layers

Lines and text components in the input data are ignored by the DRC. Even when they are present on a layer used as both an input layer and an output layer, they will be stripped from the input data and will not be present in the output data.

Forcing the DRC to handle shapes dangerously can result in shapes in subcells to not be merged. See page 138.

One of the first preprocessing steps the DRC performs on your layout data is to convert all shapes, including wires, to polygons. All touching polygons on the same layer in the same panel are then merged into single polygons. This is done before any rules are processed.

Only shapes in the current panel (and touching shapes on the same layer in neighboring panels) are merged. To learn more about panels, see page 118.

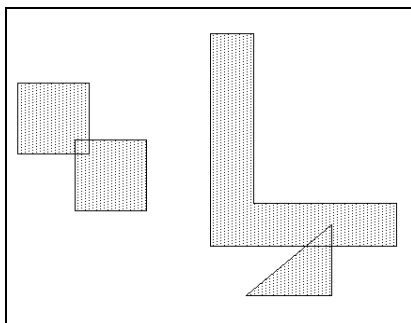


Figure 31: Layer before DRC preprocessing.

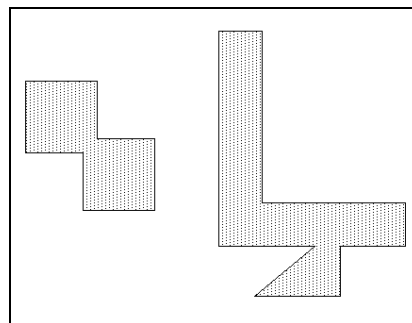


Figure 32: Layer after DRC preprocessing.

The topology of the original shapes is not used by the DRC. The program will use the topology and dimensions of the merged shapes.

This also means that wire shapes copied to (or used to generate) an output layer will be created as polygons in the output data.

IN_CELL Processing

There are three ways to force the DRC to handle shapes in certain cells differently than other shapes on the same layer. You may want to do this because the shapes in certain cells should be considered devices, even though other shapes on the same layer are considered to be conductive material.

Inductors are a good example of this problem. If you have a winding shape on a metal layer that you consider to be an inductor, you may want this metal to be on a different DRC layer and test this layer with different rules. If you put this shape in a separate cell it is relatively easy to do this.

There are three ways to filter a layer based on the cell in which it is contained:

INPUT LAYER rule IN_CELL parameter This method will place all shapes on a layer contained in a specific subcell on a different layer during input processing.

IN_CELL rule This method allows you to filter any layer contained in a cell **and its subcells** at any time in the rule set.

Layer 0 processing Layer 0 represents the bounding box of a cell. If you use the INPUT LAYER IN_CELL parameter to define layer 0, you will store a rectangle that covers all shapes in the cell. You can then use Boolean processing to filter all shapes within the rectangle. In this case, all shapes within the rectangle are processed differently, regardless of the cell that contains them.

Subject	Description	Page
INPUT LAYER rule	Define input layers	217
IN_CELL	Classify shapes in certain cells	215
Example of Layer 0 processing	Classify flattened shapes in all cells within a specific cell boundary.	221

Figure 33: References for IN_CELL processing

Types of DRC Layers

In the DRC database, all layers have two important properties:

Geometric basis:	polygon layer vs. wire layer
Error status:	error layer vs. non-error layer

All input layers are polygon layers. (Remember that DRC preprocessing will convert all wire components into polygons.) All layers that are used to generate mask layers should be polygon layers.

Wire layers are generated by DRC verification rules to mark edges as errors. Any layers generated by rules that create wires (as indicated in the table on the next page) cannot be used on right side of the '=' in any succeeding rule. You cannot perform Boolean or any other processing on wire shapes.

You can use the OUTPUT LAYER rule to define a layer that would normally be a polygon layer as a wire layer. You may want to do this because you consider shapes on that layer to be errors and you want all of your error layers to have similar properties.

When you use the WIRE keyword in the OUTPUT LAYER rule for a layer that contains polygons, the shapes on the layer remain polygons during the DRC processing. Only during output file creation are the shapes converted to wires that outline the boundaries of the polygons.

Any output layer can be defined as an error layer. Only shapes on error layers are added to the error count. Some rules (see table on the next page) automatically classify their result-layers as error layers. Other layers that may be the result of Boolean or other processing must be defined explicitly as error layers in the OUTPUT LAYER rule if you want shapes on that layer to be included in the error count.

How the DRC Works: Layer Processing

Rule	Use	Geometric basis	Error status
AND	Boolean AND of two layers	Polygon	Non-error
ASPECT_RATIO	Classify shapes by relative dimensions	Polygon	Non-error
Assignment Rule / NOT	Copy layer or inverse of layer	Polygon	Non-error
BLOAT	Expand shapes	Polygon	Non-error
BOUNDS	Classify shapes by size	Polygon	Non-error
BRIDGE	Recognize air bridges	Polygon	Non-error
HOLE_AREA- _FRACTION	Classify polygons with holes	Polygon	Non-error
IN_CELL	Classify shapes in certain cells	Polygon	Non-error
IS_BOX	Classify rectangles by size	Polygon	Non-error
IS_CIRCLE	Classify circular polygons	Polygon	Non-error
ISLANDS	Find holes	Polygon	Non-error
MAX_ANGLE	Find sharp points in notches	Wire	Error
MAX_SPACING	Classify shapes by distance apart	Polygon	Non-error
MIN_ANGLE	Find sharp points in protrusions	Wire	Error
MIN_AREA	Find small shapes	Polygon	Error
MIN_NOTCH	Find small notches	Wire	Error
MIN_SIDE	Find shapes with at least one small side	Wire	Error
MIN_SPACING	Find shapes too close together	Wire	Error
MIN_WIDTH	Find shapes with small width	Wire	Error
OFF_GRID	Find vertices not on resolution grid	Polygon	Error
OR	Boolean OR of two layers	Polygon	Non-error
OVERLAPPING	Find shapes with common area	Polygon	Non-error
SHRINK	Shrink shapes uniformly	Polygon	Non-error
SNAP	Relocate vertices on resolution grid	Polygon	Non-error
SNAP45	Relocate vertices on resolution grid preserving slope of 45° angles	Polygon	Non-error
STAMP	Find improperly connected wells	Polygon	Error
TOUCHING	Find touching shapes on different layers	Polygon	Non-error
WARN_ACUTE	Identify all acute angles on output layers	Wire	Non-error
XOR	Boolean exclusive OR	Polygon	Non-error

Figure 34: Properties of layers generated by DRC rules

Layer Generation Rules

Layer generation rules create polygons on output or scratch layers based on the contents of existing layers.

These rules are used to manipulate layers for several reasons:

Layers used to design the layout can be transformed automatically into mask layers for export.

Layers may need to be combined into mask layers to test design rules.

Certain shapes on a layer may need to be filtered into subsets to test design rules that depend on certain properties or to avoid false errors.

Some classes of design errors are found not by spacing or other verification rules, but by simple Boolean processing (e.g. ERR1= VIA AND NOT M1).

The *result_layer* of a layer generation rule will always be cleared of its previous contents and replaced with the result of the operation. If the previous contents have not been used yet in another rule, the rules compiler will warn you. The *result_layer* can be the same layer as one of the layers to the right of the '='. The following is a valid rule:

Example: **SUBSTRATE = SUBSTRATE AND NOT PWELL**

Boolean Processing

Several Boolean operations cannot be combined into a single rule (other than the use of the NOT keyword). Complex Boolean processing must be broken down into separate rules.

Example:

POLY_IN =	POLY_WIRES	OR	DEV_POLY
RESISTOR_POLY =	POLY_IN	AND	RESISTOR_MASK
POLY =	POLY_IN	AND NOT	RESISTOR_MASK

How the DRC Works: Layer Processing

Parentheses are **not** allowed in Boolean expressions. "C = (NOT A) OR B" may seem like the natural way to write a rule, but it will generate syntax errors. In a DRC rule, the NOT keyword always applies only to the layer it precedes.

Example: **C = NOT A AND NOT B**

This rule will be interpreted by the DRC compiler as:

C = (NOT A) AND (NOT B)

You must add the ERROR keyword to the OUTPUT LAYER rule when you want to add shapes generated by these rules to the error count.

Rule	Use	Page number
AND	Boolean AND of two layers	183
Assignment Rule / NOT	Copy layer or inverse of layer	187
OR	Boolean OR of two layers	283
XOR	Boolean exclusive OR	313

Figure 35: Boolean layer generation rules

Classifying Shapes by Size or Shape

Several rules classify shapes by size. The IS_CIRCLE rule finds circular polygons in arbitrary size ranges. The IS_BOX rule is used to filter rectangles by size.

The BOUNDS rule is very similar to IS_BOX, but the size criteria apply to the size of the bounding boxes of shapes. The bounding box of a shape is the smallest rectangle square with the axes (i.e. a rectangle with horizontal and vertical sides) that encloses the shape. The ASPECT_RATIO rule classifies shapes by the ratio of the dimensions of their bounding boxes. Since the BOUNDS and ASPECT_RATIO rules classify shapes by the size of their bounding boxes, they are useful to classify non-rectangular shapes.

However, when the shapes you need to classify by size are long irregular shapes, like wires, none of the rules above will be of much use. Let us say that you need to apply different minimum distance rules depending on the width of wires. Wires that are 2 microns wide must be at least 2 microns apart; however wires that are only 1.5 microns wide must be only 1.5 microns apart. The best way to classify wires by width is to shrink the wires by half the width, then bloat them again by the same amount. Wires narrower than the width will disappear during the shrink. The bloat returns the other wires to their original size.

Example:

```
M1_SHRINK = SHRINK (M1_IN, .999)
M1_2_WIDE = BLOAT (M1_SHRINK, .999)
M1_UNDER_2 = M1_IN AND NOT M1_2_WIDE

ERR1 = MIN_SPACING (M1_2_WIDE, M1_2_WIDE, 2)
ERR2 = MIN_SPACING (M1_2_WIDE, M1_UNDER_2, 2)
ERR3 = MIN_SPACING (M1_UNDER_2, M1_UNDER_2, 1.5)
```

Note that the value used to separate the wires is **slightly less** than half of the size criteria. If the SHRINK and BLOAT rules used a value of exactly 1.0, then wires 2 microns wide would wind up on the M1_UNDER_2 layer. These rules separate M1_IN shapes into those wider than 1.998 and those 1.998 wide or narrower. Each of these subsets is then verified for different minimum spacing. Note that the distance between shapes in either subset is also checked with the rule that generates the ERR2 layer.

The SHRINK and BLOAT rules are fairly expensive in terms of processing time, and they can lead to the distortion of polygons with varying width or skewed sides. A single polygon with a thin segment in the middle may wind up as two polygons after a shrink and bloat. Shrinking then bloating a shape with a skewed side (i.e. a side that is not horizontal or vertical) may result in the slope of the skewed side changing because the vertices of such sides are often not on grid after the shrink. The bloat operation then magnifies the problem. Look carefully at the layers created before you rely on them for design rules checking.

If the M1_IN layer contains acute angles, you may want to add a pair of BLOAT_ANGLE rules around the BLOAT rule to prevent the acute angles from being cut by that rule. However, decreasing the bloat angle will increase the reach of the rule and the DRC processing time. See the description of the BLOAT_ANGLE rule for details.

How the DRC Works: Layer Processing

The MIN_AREA rule is usually used as a verification rule to find shapes that violate a design restriction on the minimum area of shapes. However, it can be used as a filter to classify shapes by size. Let say that you need to apply different verification rules to shapes with an area larger than a certain minimum size. The following rules will filter shapes based on area without classifying shapes with a small area as errors.

Example:

```
INPUT LAYER           1 A  
OUTPUT ERROR LAYER   11 ERR1  
OUTPUT LAYER         0 SMALL_A; 0 LARGE_A
```

```
SMALL_A=   MIN_AREA   (A, 10/BORDER=0)  
LARGE_A=   A AND NOT SMALL_A  
ERR1=      MIN_WIDTH  (LARGE_A, 3)
```

Since the SMALL_A layer is defined with layer number 0, shapes created on that layer will not automatically be counted as errors as they would normally be.

The table on page 62 lists the geometric basis and automatic error status of shapes created by each rule.

The MIN_AREA rule can be used as a filter this way because it generates shapes with a polygon geometric basis. The DRC can use these shapes in other rules in the same manner as any other rule that generates polygons. The shapes created by verification rules that generate wires cannot be used this way. Layers that contain wire shapes cannot be used on the right side of the '=' in any other rule.

One other rule that involves classifying shapes by size is the HOLE_AREA_FRACTION rule that classifies shapes by the fraction of their area that is removed by holes.

Rule	Use	Page
ASPECT_RATIO	Classify shapes by relative dimensions	184
BLOAT	Expand shapes	189
BLOAT_ANGLE	Specify how acute angles are processed by the BLOAT and SHRINK rules	311
BOUNDS	Classify shapes by the size of their bounding boxes	194
HOLE_AREA-FRACTION	Classify shapes by fraction removed by holes	211
IS_BOX	Classify rectangles by size	222
IS_CIRCLE	Classify circular polygons	225
MIN_AREA	Isolate shapes with small area	243
SHRINK	Shrink shapes	302

Figure 36: Rules used to filter shapes by size

Classifying Shapes by Distance

The related MIN_SPACING rule finds errors rather than classifying shapes.

The MAX_SPACING rule finds shapes on a certain layer that are at least a specific distance away from other shapes on the same layer or from shapes on a different layer. You can tailor the rule to collect only shapes that are, or are not, electrically connected from the other shapes. You can also treat shapes that are near the corners of each shape differently from those that are near a side.

This rule can also be written to collect only shapes that are within the indicated distance from the other shapes.

Rule	Use	Page
MAX_SPACING	Classify shapes by distance to other shapes	235

Overview of Other Layer Generation Rules

The BLOAT and SHRINK rules are also used to expand or reduce shapes on a layer for reasons other than separating shapes on a layer by size (covered on page 65.) These rules can be used to change a layer used to design the layout to mask layer dimensions before design rules are applied. They can also be used in sophisticated processing to automatically generate a mask layer from layout layers. (See the example on page 71.) The BLOAT_ANGLE rule changes how acute angles are handled for both rules.

The OVERLAPPING and TOUCHING rules classify shapes by whether or not they touch shapes on a second layer. The OVERLAPPING rule will recognize shapes that share a common area. The less restrictive TOUCHING rule will identify shapes that share area or just a line segment along two sides. This can be very useful in many situations. These rules are commonly used to identify devices. They can also be used to verify that non-design shapes (often referred to as dummy shapes) are located in the correct place. A TOUCHING rule is often used in place of a Boolean rule to test that two layers do not overlap since it will locate shapes that share an edge.

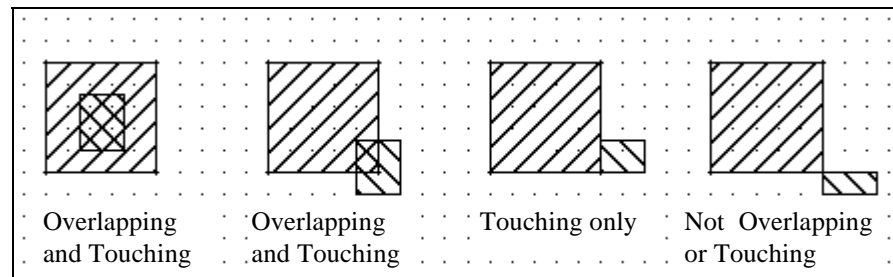


Figure 37

The ISLANDS rule is used to find unconnected shapes on a layer that should be composed solely of a single connected shape. This rule is more commonly used to find holes in large shapes that span your layout. The HOLE_AREA_FRACTION rule finds shapes that have at least a minimum area removed by holes. The BRIDGE rule finds air bridges. (This rule is of interest primarily to users of the Gallium Arsenide technology.)

You must add the ERROR keyword to the OUTPUT LAYER rule when you want to add shapes generated by these rules to the error count.

Rule	Use	Page
BLOAT	Expand shapes	189
BLOAT_ANGLE	Specify how acute angles are processed by the BLOAT and SHRINK rules	311
BRIDGE	Recognize air bridges	196
HOLE_AREA-FRACTION	Classify shapes by fraction removed by holes	211
ISLANDS	Find holes or unconnected shapes	230
OVERLAPPING	Find shapes on different layers with common area	288
SHRINK	Shrink shapes	302
TOUCHING	Find touching shapes on different layers	311

Figure 38: Other non-error layer generation rules

Generating Output Layers

One of the last tasks performed at the end of a DRC run is to export shapes to a command file. This command file can create shapes in the ICED™ layout editor. All shapes on non-error output layers are created in same command file as error shapes (unless the `HIERARCHICAL` command line option is used. See page 146.)

Unless the `HIERARCHICAL` command line option is used, all shapes will be created without cell hierarchy in the current cell when the command file is executed in the layout editor. If the current cell already has shapes on the same layer numbers as those used as output layer numbers in the DRC rule set, you can corrupt the contents of the current cell. There is no easy way to classify the shapes just added to your cell from the original contents unless you have added all shapes to new layer numbers.

If you are using the DRC to create layers for import to your existing cells, you can use unique layer numbers in the `OUTPUT LAYER` rule, then change the layer number of these shapes later in the layout editor. The `SWAP` command in the layout editor will change the layer number of selected shapes.

To learn how to import shapes on output layers into your design, read about the DRC command file beginning on page 365. The following pages describe important issues about how the DRC generates shapes on output layers. This information is primarily of interest to people who will be using the DRC to generate mask layers for import back into their design.

Example of Generation of P-Select and Diffusion Mask Layers

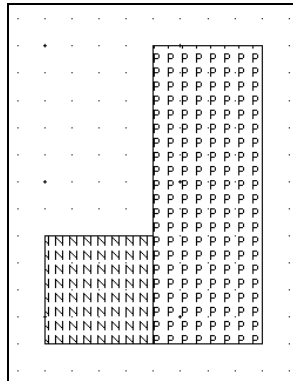


Figure 39: N and P shapes for NP2DS examples.

Let us cover an example of generating mask layers from design layers. In this example, the layout has been created with the N and P layers for the bulk or well layers for transistors. These layers must be transformed into the DIFF (for diffusion) and PSEL (for P-select) layers for mask processing. The DIFF layer is the union of both the N and P layers, while the PSEL layer is a bloated area around the P shape that transforms the DIFF layer into a P-well. A PSEL shape should never overlap an N shape, or the well will wind up as a partial P-well rather than an N-well.

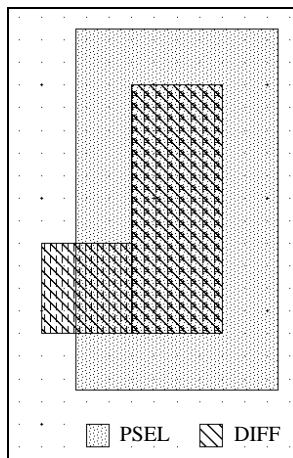


Figure 41: DIFF and PSEL layers generated by NP2DS0 rules

A simple attempt at the rules to create the DIFF and PWELL layers is shown in Figure 40. This rule set will work acceptably on most N and P shapes, however, when N and P shapes touch, there is a problem. The result of these rules is shown in Figure 41. Note that the PSEL layer overlaps the N-well shape. This will result in an incorrect mask set.

```
input layer{    2 n;
               3 p;
}
output layer{  42 diff;
               43 psel;
}
diff = n or p;
psel = bloat(p, 2.5);
```

Figure 40: NP2DS0 rule set

A slightly better version of the rule set is shown in Figure 42. This version generates a temporary layer P1 that has a bloated N layer (temporary layer N1) subtracted from it. The results are shown in Figure 43. Now the PSEL shape does not overlap the N shape. However, these shapes share a horizontal edge. If the mask sets are slightly misaligned during mask processing, these shapes will overlap.

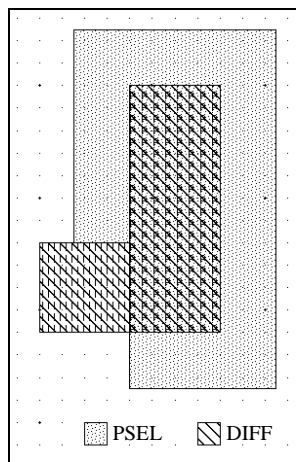


Figure 43: DIFF and PSEL layers generated by NP2DS1 rules

```
input layer{    2 n;  
               3 p;  
}  
output layer{  42 diff;  
              43 psel;  
              102 n1;  
              103 p1;  
}  
diff = n or p;  
  
n1 = bloat(n, 2.5);  
p1 = p and not n1;  
psel = bloat(p1, 2.5);
```

Figure 42: NP2DS1 rule set

The DRC can perform more processing on the N and P layers to minimize this problem, resulting in a less hazardous mask set.

Look at the rule set in Figure 44. This rule set performs a series of bloats and subtractions when building the PSEL layer. This results in a diagonal edge on the PSEL shape that minimizes mask misalignment problems.

This rule set could still be improved by adding rules that test for the overlap of the N and PSEL layers. Also the CUT_RESOLUTION rule should be added to avoid potential mask problems. These and other DRC output layer post-processing issues are covered next.

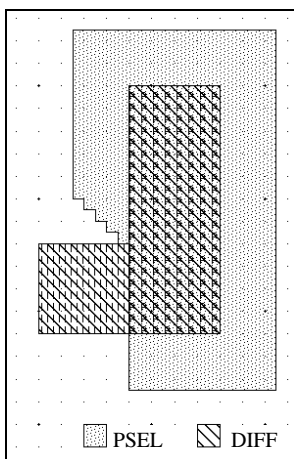


Figure 45: DIFF and PSEL layers generated by NP2DS2 rules

```

input layer{      2 n;
                  3 p;
}
output layer{    42 diff;
                  43 psel;
output layer{    0 n1;
                  0 p1;
}!these are reusable scratch
layers

diff = n or p;

n1 = bloat(n, 2.5);
p1 = p and not n1;
psel = bloat(p1, 2.5);

p1 = bloat(p, 0.5);
n1 = bloat(n, 0.5);
n1 = n1 and not p1;
psel = psel and not n1;

p1 = bloat(p, 1.0);
n1 = bloat(n, 1.0);
n1 = n1 and not p1;
psel = psel and not n1;

p1 = bloat(p, 1.5);
n1 = bloat(n, 1.5);
n1 = n1 and not p1;
psel = psel and not n1;

p1 = bloat(p, 2.0);
n1 = bloat(n, 2.0);
n1 = n1 and not p1;
psel = psel and not n1;

```

Figure 44: NP2DS2 rule set

Problem Shapes for Mask Generation

There are two special classes of polygons that are likely to cause problems for mask processing software: bad polygons and acute angles. If your design contains these types of shapes, they should be fixed before your final design.

Bad Polygons

Bad polygons can be present in your input data, they will never be generated by the DRC. The DRC will by default locate all bad polygons on input layers during preprocessing.

The simple definition of a bad polygon is a polygon with self-intersecting sides that are not adjacent. Let us visualize drawing a "good" polygon on a piece of paper. As your pencil draws the sides of the shape in a clockwise direction, the inside area of the polygon should always be on the right. Your pencil should never cross an existing side. The following types of polygons are all "bad".

All input layers are checked for bad polygons by default. For this reason, it is a good idea to define all mask layers as input layers, even if they are not verified by any rules.

"Bow tie" shapes: An example of this type of bad polygon is shown in Figure 46. Note that as you trace the diagonal sides, the inside area of the polygon shifts from the right side of the line to the left side. Mask processing software will usually fail to generate the shape you would expect from this type of data. You should draw this type of shape as two properly drawn triangles that touch at a point.

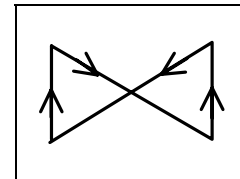


Figure 46: "Bow tie" bad polygon

Improperly drawn holes: When you create a polygon with a hole in ICED™, the sides of the hole must be connected to the sides on the outside of the polygon. If the sides cross so that the edges of the hole and the outside edges are both drawn in a clockwise direction, the shape cannot be processed properly. If you visualize drawing this shape, you can see that the inside area of the shape shifts from the right of each edge to the left of the hole edges.

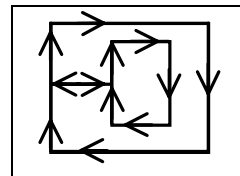


Figure 47: Improperly drawn hole

Self-intersecting shapes: Letters are a common case of this type of bad polygon. When the interior area of the polygon intersects itself and creates a hole, most mask processing software will not create the shape you expect. Draw this type of shape without intersecting sides.

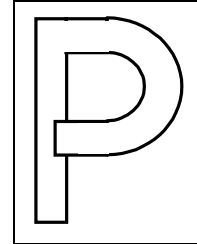


Figure 48:
Self-intersecting shape

Bad polygon shapes found as the input data is processed will be reported in the log file, then copied to a special error layer. This special layer is layer number 99 by default. This layer number can be changed to a different number with the BAD_POLY rule.

Bad polygon shapes are not added to the error count. When bad polygons are present, the console messages and the log file will both report warnings, but the error count will not be incremented.

The copied bad polygon shapes are created in the subcell error command files rather than the main DRC command file. These files have a file name extension of .ERR. One subcell error command file is created for each subcell containing errors that can be found without flattening the data. One bad polygon in a subcell used 100 times in the main cell will result in one error message, not 100 error messages. One shape will be created in the corresponding subcell error command file. The shape is created in the coordinates of the subcell. This enables you to locate and correct the error while editing the subcell.

If you use the BAD_POLY rule to set the bad polygon layer number to 0, the creation of shapes in the subcell error command files is suppressed. No shapes will be created to mark the error. However, the log file will still contain warning messages about bad polygons with the cell names and coordinates indicated.

By default, the DRC will locate bad polygons on all input layers defined in the rule set. If you prefer to have the DRC ignore input layers that are not actually used in other rules, add the NO_CHECK_INPUT rule to your rule set.

How the DRC Works: Layer Processing

All bad polygon shapes remain in the DRC database during the run. They will be used in layer generation and verification rules, however, a shape with an improperly drawn hole will be handled as though the hole is not there.

Acute Angles

The
WARN_ACUTE
rule can be used
to change the
layer number of
the acute angle
error marks.

By default, the DRC marks on layer number 99 all acute angles on all **output** layers. This test is performed at the end of the DRC run when shapes are created in the output files. Shapes with acute angles that are not on output layers will not be found. If the NO_WARN_ACUTE rule is present in the rule set, this test will not be performed.

This test is intended primarily to find acute angles that have been created inadvertently by the DRC on mask layers. There are two cases where shapes with acute angles may be created from shapes that have no acute angles:

- 1) A horizontal or vertical panel boundary cuts a shape with skewed sides.
- 2) A polygon with holes, or more than 199 vertices, has been cut into multiple shapes to translate it into valid ICED™ polygons.

Both of these types of problems can be fixed by hand in the ICED™ layout editor.

An example of case 1 is shown in Figure 49. The polygon has been cut into two polygons by a vertical panel boundary. The new polygon on the left has two acute angles. These acute angles have been marked with wires by the DRC. This type of problem is solved by joining the two polygons again using the MERGE command in the ICED™ layout editor.

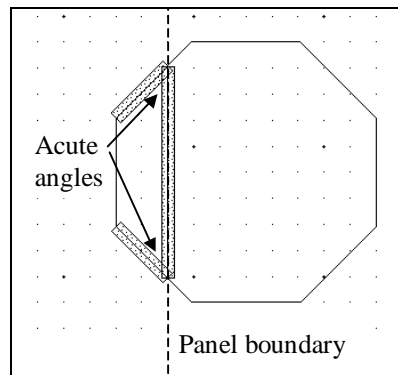


Figure 49: Polygon cut by panel boundary.

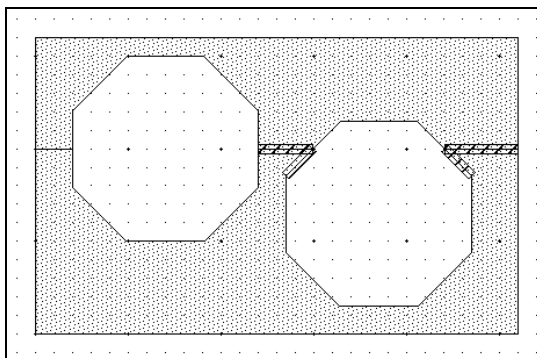


Figure 50: Polygon with holes cut by DRC to be valid ICED™ shapes.

An example of case 2 is shown in Figure 50. A polygon with holes has been created through layer processing in the DRC run. The DRC must cut this shape into two valid ICED™ shapes. The bottom polygon has two acute angles marked with error wires by the DRC.

This type of problem may require you to cut the polygons at other locations (using the CUT command in the ICED™ layout editor) and join the polygons (using the MERGE command) in a different combination to avoid the sharp angles. In the case of Figure 50, two vertical cuts and two merges will allow you to create polygons without acute angles.

Both acute protrusions and acute notches will be marked by the acute angle test.

The error wires are created on layer number 99 or the layer number specified in the WARN_ACUTE rule. The wires are created in the main output command file.

The acute angle errors are not added to the DRC error count. However, error messages about the presence of acute angles found by the DRC are shown in the console log and in the DRC log file near the error count.

Post-processing of Output Layers

The DRC must perform some post-processing of shapes on generated layers that will cut shapes or distort their shape slightly. This is an unavoidable result of grid resolution issues and panel processing.

Some shapes cannot be represented in the ICED™ editor as simple polygons. For example, the ICED™ editor will not allow shapes with holes or shapes with more than 199 vertices. However, the DRC does **not** have these limitations. When the DRC creates polygons like these, they are used by other rules in their true shape. It is only at output that these shapes are modified to be valid ICED™ polygons.

See other examples of this problem on page 131.

Shapes created by the DRC that cross panel boundaries are broken at the panel boundaries during the DRC run. If a shape crosses the vertical or horizontal panel boundary at a skewed angle, there may be a tiny displacement of the vertex coordinates where the shape is cut by the boundary to keep the vertices on grid. Shapes that have been cut at the panel boundaries are not merged before output. This can occasionally lead to the acute angles on output layers problem we have just covered. These cuts can also lead to vertex resolution problems.

The resolution grid used by the DRC is much finer than that used by the ICED™ layout editor. Shapes on output layers may have their vertices shifted to lie on the grid used by the layout editor.

Resolution Grids

There are two different resolution grids involved when using the DRC. A resolution grid is defined as valid points a certain integral number of units away in the x or y directions from an origin point. The coordinate data for every vertex must be expressed in terms of points on the resolution grid.

The resolution grid used by the DRC is much finer than the resolution grid used by the ICED™ layout editor. The DRC resolution grid is usually 16 times finer, but very large designs may require a coarser grid. This allows the DRC to resolve the results of intersections much more accurately.

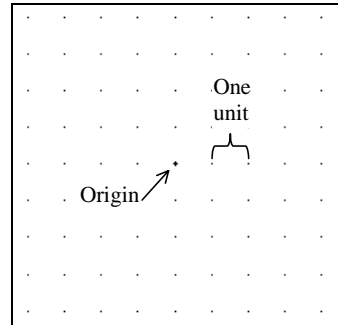


Figure 51: Resolution grid

Since the DRC uses a finer grid than the ICED™ layout editor, shapes created by bloats, shrinks, or intersections of skewed sides in the DRC may have vertices that are not on the ICED™ grid. When these shapes are created in the output data, they must be snapped to the nearest point on the ICED™ grid.

For example, let us say that the ICED™ grid has a .0001 user unit resolution. (This is a typical value.) If your design rules call for a bloat smaller than .0001, say .00005, the DRC can perform this operation with valid results. However, when you output this layer, the vertices must be snapped to the ICED™ grid. Some sides will wind up shifted by .0001, other sides will be shifted back to their original location before the bloat.

Another example of the different resolution grids is when a shape with a skewed side (a side that is neither vertical nor horizontal) is cut in two by a panel boundary. The exact location of the new vertices must be rounded to lie on the DRC resolution grid. This can lead to a bend in the skewed side.

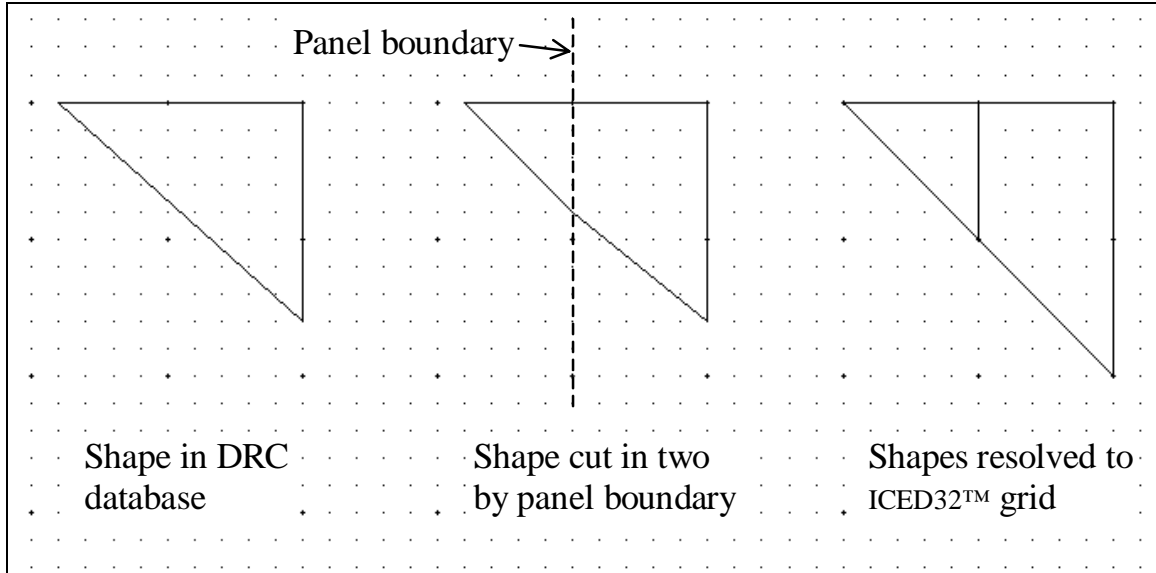


Figure 52: Shape with skewed side cut by a panel boundary then resolved to ICED™ grid

Look at Figure 52. Let us say that the small dots represent the DRC resolution grid and that the crosses represent the ICED™ grid. When the shape on the left with the skewed side is cut by a vertical panel boundary, the y-coordinates of the new vertices must be rounded to lie on the DRC grid. This has led to a bend in the skewed side. This bend will remain in the data during the DRC run. When the shapes are output as ICED™ data, the vertices are resolved to lie on the ICED™ grid. In this case, resolving the data to the courser grid has removed the bend. In some cases, resolving the data to the new grid will make the problem worse.

Now let us suppose that the bend has resulted in errors from verification rules. When you look at the output data, you see no bend. The fact that the shapes that are checked are not the same shapes in the output data can lead to some confusing errors. This is a result of the greater accuracy of the DRC data than the layout data.

Another example of this type of problem is a shape generated during the DRC run that is very thin sliver. If this sliver is less than one ICED™ database unit wide, the sliver will disappear in the output data. If the sliver violated minimum width or minimum spacing rule checks, it can be difficult to determine why the error is there.

One way to avoid these mysterious errors is to resolve shapes on the problem layers to the ICED™ grid before you test them with design rules. The SNAP and SNAP45 rules can be used for this purpose.

The SNAP and SNAP45 rules can be used to snap vertex data to any desired grid. This can result in the distortion of shapes. Portions of shapes that are collapsed to zero width will disappear without warning. Sides that are not horizontal or vertical will often have their slope changed somewhat. The SNAP45 rule differs from the SNAP rule in that sides at 45° angles will have their slope preserved. This often requires the addition of vertices that add ledges or cut off corners.

Shapes that are cut by the DRC to create valid ICED™ polygons, or that are cut at panel boundaries, may have off-grid vertices. (See an example on page 77.) You can prevent this problem with the optional CUT_RESOLUTION rule.

One last rule that can help you find grid resolution problems is the OFF_GRID rule. This rule will mark as errors all polygons with vertices that are not on a specific grid.

Keep in mind that touching shapes are merged by the DRC before any of these rules are executed. If touching shapes share off-grid edges that disappear when the shapes are merged, the merged shape may have no off-grid edges. In this case, the OFF_GRID, SNAP, and SNAP45 rules **will not** flag the merged shape as having off-grid vertices.

Recommended Procedure for Writing Rules to Generate Mask Layers

Use the following checklist when generating mask layers:

Write a simple version of the rules first to test the process.

Run a test case using a subset of your design to debug and revise the rule set until you are generating the layers you require.

Add SNAP or SNAP45 rules to resolve shapes on new layers to the ICED™ grid before testing them with verification rules.

Add DRC design rule verification checks for the new mask layers to insure that no violations are being generated.

Add the CUT_RESOLUTION rule to force cut lines to be on the ICED™ grid.

Insure that no NO_WARN_ACUTE rule is in the rule set. Remove the BAD_POLY=0 rule if it is present in your rule set.

Run the rule set on your whole chip and carefully inspect the results for unusual cases you may not have considered when writing the rule set.

Always use the SLOW option on the DRC program command line to insure correct layer generation.

If you will be generating mask shapes from a nested design:

Read about hierarchical output to create the DRC output data in nested cells that can be imported into each of your original cells.

Read about dangerous processing to generate more shapes in nested cells. Be sure to carefully look at and resolve any DANGER warnings in the log file. (See page 136.)

Subject	Importance	Page
Grid resolution	Issues important to placement of vertices on a grid acceptable to mask processing software	79
Panel processing	Issues important to why shapes are cut at panel boundaries	118
Hierarchical processing	Method to follow if you want to generate output data with the nested cell structure of the input data	146
Example of hierarchical processing	Detailed example of generating a layer hierarchically	418
BAD_POLY rule	Sets the layer number for shapes that mark bad polygons on input layers	189
CUT_RESOLUTION rule	Define grid for shapes cut by the DRC	205
NO_CHECK_INPUT	Avoid marking bad polygons on unused layers	276
OFF_GRID rule	Classify as errors shapes that have vertices off of the specified grid	282
SNAP rule	Relocate vertices on resolution grid	304
SNAP45 rule	Relocate vertices on resolution grid preserving slope of 45° angles	306
WARN_ACUTE rule	Determines output layer for acute angle markers	313
NO_WARN_ACUTE rule	Disables acute angle test	280
DRC command file	Details on how to import shapes on output layers into layout editor.	365
Subcell command files	Command files that create shapes marking BAD_POLY errors in subcells	375

Figure 53: References for output layer generation issues

Spacing Verification

The MAX-
_SPACING
rules classifies
shapes by how
far away they
are from other
shapes rather
than marking
errors.

Spacing errors between shapes are checked with the MIN_SPACING rule.

The MIN_SPACING rule is one of the DRC rules that checks and marks sides of shapes rather than entire shapes. Violations of this rule create wire shapes that mark the portions of sides that are too close to each other. These violations are automatically added to the error count.

The DRC minimum spacing algorithm checks the distance between the vertices of one shape to the sides of other shapes. If no vertex is too close to a side, no error will be found.

When shapes do not overlap, minimum spacing violations are always found. Unless the QUICK_SPACING option is used on the DRC command line, there is no way for two non-overlapping shapes to be within a minimum distance of each other and not have a vertex be within the minimum distance. (We will cover the QUICK_SPACING option a little later on page 100.)

When shapes do overlap, it is common sense to assume that they violate a minimum distance rule, since they touch. However, since the MIN_SPACING rule uses vertex data, overlapping shapes may not violate the rule. Two sides can cross each other with no vertex within the minimum distance.

There are two cases of this quirk of minimum spacing:

Enclosure: one shape is covered by another.

Crossing shapes: one shape crosses another.

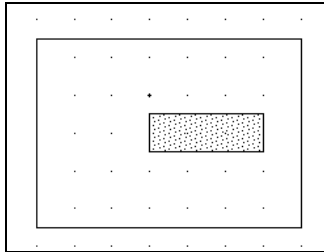


Figure 54: One shape enclosing another

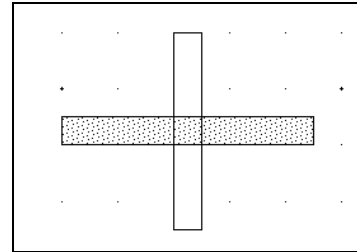


Figure 55: Crossing shapes

Using Rules Other Than MIN_SPACING to Mark Spacing Problems

Overlaps and enclosed shapes

Unless a vertex of one shape is within the minimum distance of a side of the other, shapes like those above will not be marked. Fortunately, when overlapping shapes are always considered minimum space violations, there is an easy way to find them. Use the AND rule. The AND rule will mark all cases where shapes on two layers overlap with a non-zero common area.

Example: **OUTPUT ERROR LAYER 101 ABCROSS**

ABCROSS = A AND B

The AND rule above will create shapes on layer ABCROSS wherever a shape on layer A overlaps a shapes on layer B with non-zero area. These shapes will be counted as errors since the ERROR keyword is used in the layer definition statement for layer ABCROSS. **If the ERROR keyword was not used, shapes on ABCROSS would not be counted as errors.**

How the DRC Works: Spacing Verification

If overlaps are not always errors, you may need to use other rules to find errors when shapes can overlap.

Let us say that enclosure of shapes on one layer by the other is acceptable. However, shapes like the one in Figure 56 are not. A MIN_SPACING rule will not mark this relationship as an error unless a vertex of one shape is too close to a side of the other. However, the following Boolean rule will place on layer ERR all shapes on layer A that are not enclosed by layer B.

Example: **ERR = A AND NOT B**

Remember to define layer ERR as an error layer so that shapes on that layer are counted as errors.

By default, coincident edges are considered violations by the MINSPACING rule. However, you can inadvertently prevent them from being found by adding keywords to the rule. The Boolean rules above will not find errors like coincident edges. When incomplete enclosure with a coincident edge (as seen in Figure 57) is a violation, it is highly recommend to add rules similar to the following to find problems like this and the one shown in Figure 56.

Example: **NOTB = NOT B**
ERR = A TOUCHING NOTB

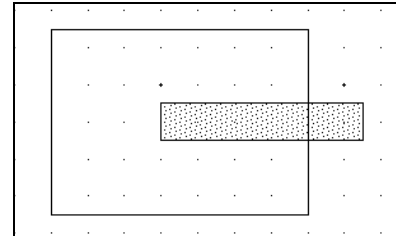


Figure 56: Overlapping shapes that may not be marked by MIN_SPACING rule.

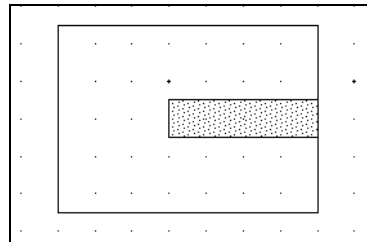


Figure 57: Incomplete enclosure.

Let us make this example a little more complicated by saying that layer A is contact layer that is valid when covered by a shape on either layer C or layer B. So the example above will mark many valid A layer contacts to layer C as errors. To avoid marking layer C contacts as false errors, rewrite the rules as follows:

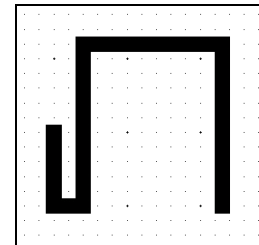
Example:

$$\begin{aligned} \text{BC} &= \text{B OR C} \\ \text{NOTBC} &= \text{NOT BC} \\ \text{ERR} &= \text{A TOUCHING NOTBC} \end{aligned}$$

Notches in serpentine or fingered shapes

An overview on notches is covered on page 103.

The MIN_NOTCH rule can be a very important addition to a MIN_SPACING rule when you need to find spacing errors between shapes on the same layer. Consider Figure 58. The long wire folds back on itself and two sides are very close each other. This is a notch in a single shape rather than a spacing error between shapes.



A MIN_SPACING rule will **not** mark this as an error. If your design rules consider this an error, you should add a MIN_NOTCH rule to find such errors.

Figure 58: Notch, not MINSPACING error.

Remember that all touching shapes on a single layer are merged during DRC preprocessing. So even if a spacing problem like the one in Figure 58 is caused by two separate wires on the same layer, to the DRC it will be a single shape with a notch rather than a MIN_SPACING error.

Problems similar to the one above are common in complex chips where many pieces of wire are merged into single polygons before the DRC performs a spacing check. If the DRC seems to have missed a spacing violation of this nature, add an equivalent MIN_NOTCH rule and retest.

Simple Spacing Checks

Now that we have covered what the MIN_SPACING rule does not check, we will go on to what the rule does check.

***error_layer* = MIN_SPACING (*from_layer*, *to_layer*, *distance*)**

This is the simplest form of the syntax for the MIN_SPACING rule. When written this way, all sides of shapes on the *from_layer* must be at least *distance* away from sides of shapes on the *to_layer*. Sides that are less than *distance* away will have error wires created for them on *error_layer* along the portion of the sides that are in violation. Sides that are exactly *distance* away will not be marked.

Example:

ERR = MINSPACING (A, B, 2)

This rule will verify that all sides of shapes on layer A are at least 2 user units away from sides of shapes on layer B. All portions of sides that are closer than 2 user units will be marked with error wires on layer ERR. This layer will automatically be classified as an error layer. All shapes on layer ERR will be added to the error count.

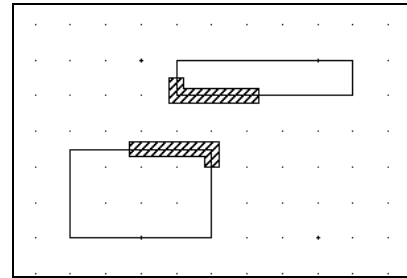


Figure 59: Error wires created for shapes closer than 2 units.

Note that the underscore ('_') can be left out of the MIN_SPACING keyword. Underscores are always optional in DRC rule keywords.

The *from_layer* can be the same as the *to_layer* in the MIN_SPACING rule. In this case any two shapes on the same layer that are too close together will be marked with errors.

Example:

ERR = MINSPACING (A, A, 2)

Optional Keywords to Reduce False Errors

Design rules often allow shapes to be closer than the minimum distance in special circumstances. Perhaps the minimum distance rule applies only if the shape on one layer is inside or outside the other layer. Perhaps only parallel sides must be the minimum distance apart and crossing sides are not violations.

See page 65 for an example of separating wires by size before applying spacing checks that depend on wire width.

If you do not write the spacing rules to take these extra criteria into account, you will see many false errors. Ignoring false errors is a dangerous thing to do. Real errors can be easily missed.

The DRC provides several keywords to narrow the search for violations. In addition you can use Boolean rules or rules that classify shapes by size or touching criteria to isolate certain shapes on a separate layer prior to applying the MIN_SPACING rule.

Directional Spacing

Adding /IN or /OUT to a MINSPACING rule can prevent coincident edges from being marked as errors. See the following examples.

Example:

The /IN and /OUT keywords are used to change a simple spacing check into a directional spacing check. When these keywords are used, the spacing criteria changes so that only sides that are found toward the inside or outside of shapes on the indicated layer are candidates for violations.

ERR=MIN_SPACING(A, B/IN, 2)

When the above rule is run on the shapes in Figure 60, only the layer A shape toward the inside of the B shape is considered for errors. The layer A shape on the outside of the B shape is ignored.

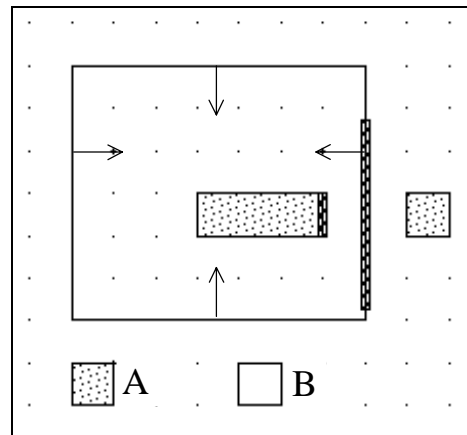


Figure 60: When B/IN is used, only shapes toward inside of B shape sides are marked.

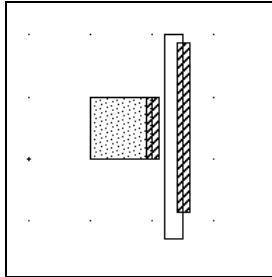


Figure 61: The shapes do not need to overlap for the /IN keyword to find errors.

When the same rule above is run on the shapes in Figure 62, you might think that both shapes with coincident edges would be marked as errors. Both boxes on layer A share an edge with the edge of the layer B shape. Therefore, the distance between these sides and the B shape side is exactly the same. However, only the top layer A shape is marked. This is because the B/IN specification means that only shapes with area toward the inside of layer B shapes are considered for violations.

Coincident edges must be considered when writing MIN_SPACING rules. If the lower shape in Figure 62 does violate the rules, the /IN specification should not be used, or another method (similar to the TOUCHING rule examples on page 86) should be used to find it.

For our next example, let us say that you need to verify that shapes on layer A that are enclosed by layer B need a 10 user unit border of layer B on all sides. However, layer A shapes that are outside of layer B shapes do not need to be this far away. Shapes on layer A that cross a layer B boundary, so that at least some non-zero area is outside of layer B should not be marked as errors.

The layout data we will use for this example is a slightly modified version of the ENCL.CEL file included with the installation. The first rule set we use is the ENCLOSUR.RUL file that is also included in the installation.

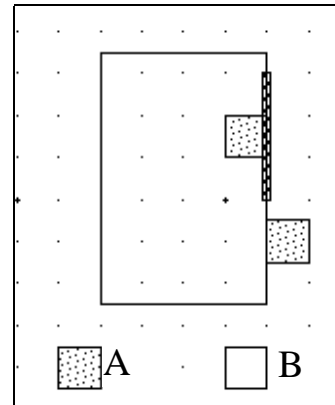
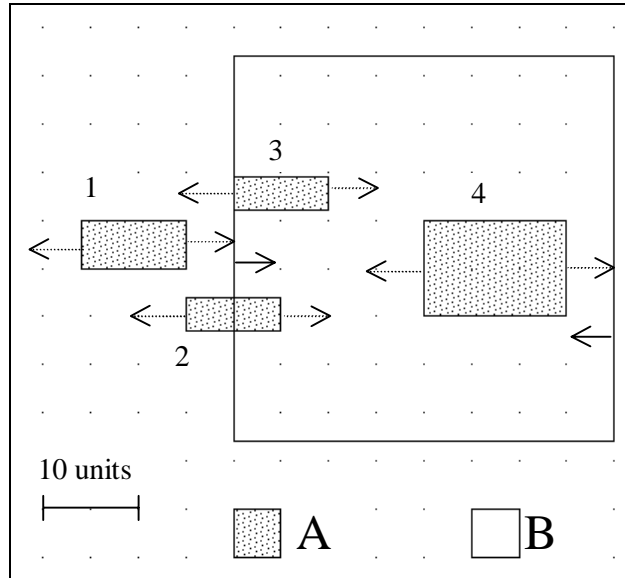


Figure 62: When B/IN is used, only shape with area toward inside of B shape is marked.

Example: **TOO_CLOSE = MINSPACING (A/OUT, B/IN, 10);**

The A/OUT specification means that only layer B sides that are found while looking out from layer A can be marked as errors. This is represented by the dashed vectors in Figure 63. The B/IN specification allows the DRC to look only toward the inside of layer B sides. This is represented by the solid vectors. (We have omitted the vertical direction for the sake of simplicity.)



Shape number 1 will not be marked because the DRC is looking for errors only toward the inside of the layer B edge. Shape number 2 will not be marked because the DRC will search for errors only looking outward from the outside edges of the shape. No edges of the layer B shape are within 10 units in either direction.

Figure 63: Modified ENCL.CEL file with arrows showing how the DRC will search for sides in violation.

Therefore, it looks like this MIN_SPACING rule will prevent the false errors we needed it to. We will test it on the layout in a moment.

In general the direction vectors for two edges must point toward each other for the DRC to find the error. We can see that the vectors point toward each other for the right edge of shape 4. The DRC will mark an error for this edge.

But how about shape 3? The direction vectors go in opposite directions, but they point away from each other, not toward each other. **The DRC will not see shape 3 as an error.**

From Figure 64 we can see that shape 3 is not marked by the MIN_SPACING rule. This is an important side effect of directional spacing verification. If you instruct the DRC to look only inside or outside of shapes for errors, **coincident edges may not be found.**

If we remove the /OUT specification from the A layer, the DRC will mark the coincident edge of shape 3, however, the right edge of shape 2 will now be marked as an error.

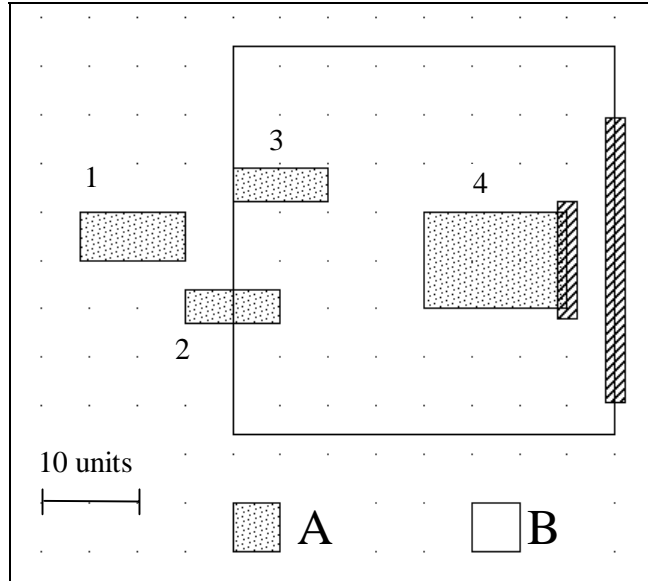


Figure 64: Modified ENCL.CEL with error marks from MINSPACING (A/OUT, B/IN, 10) rule.

(You may wonder why the crossing and perpendicular sides of shapes 2, 3, and 4 are not marked with error wires. This is due to the fact that the DRC treats these types of sides differently when directional spacing checks are performed. We will cover this later when we cover the orientation options.)

The best way to verify incomplete enclosure due to coincident edges is to add a TOUCHING rule. In this case, we want to find shapes on A that are covered by layer B, but touch area that is not layer A or layer B.

Figure 65 shows the rules we have added to the ENCLOSUR.RUL file to find incomplete enclosures due to coincident edges.

```
OUTPUT ERROR LAYER 4 COINCIDENT;
OUTPUT LAYER 0 AANDB; 0 NOT_AB;

NOT_AB = NOT A AND NOT B
AANDB = A AND B
COINCIDENT = AANDB TOUCHING NOT_AB
```

The three layer generation rules will find all shapes on layer A that are covered by layer B and touch, but do not cross, the boundary of a layer B shape.

Figure 65: Extra rules to find coincident layer A shapes.

If we used a TOUCHING rule that tested the NOT_B layer (as used in a previous example) rather than the NOT_AB layer, then shape 2 would be marked as an error as well.

Since we will find shapes like shape 3 with a TOUCHING rule, we must be sure to add the ERROR keyword to the OUTPUT LAYER rule that defines the result layer for the TOUCHING rule. Otherwise, shapes on that layer will not be counted as errors. The AANDB and NOT_AB layers are defined with layer number 0. This means that they are really scratch layers and shapes on those layers will not be included in the output command file.

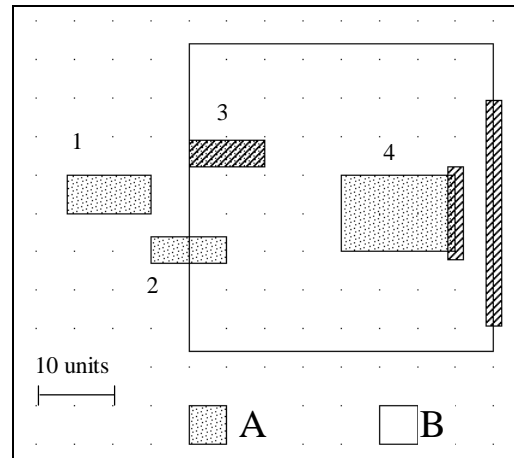


Figure 66: Modified ENCL.CEL with error marks from MINSPACING rule and TOUCHING rule.

Side-side angle exceptions– Beta test only!

New with beta version 113.65 of the DRC is the AWAY option of the MIN_SPACING rule. This option is particularly useful if wires that leave a

How the DRC Works: Spacing Verification

shape at an angle should not be marked as errors, but parallel wires should be marked if they are too close.

The AWAY option restricts errors to non-overlapping pairs of sides that are less than a certain angle apart. The AWAY option should be added to only one layer specification. We will call this layer the *away_layer* in this discussion.

When both shapes are on the same layer, the AWAY option does not apply. Refer to the MIN_NOTCH rule. Also see pages 87 and 103.

Side-side pairs that are within *distance* of each other will be **not** be marked as errors when **both** of the following conditions are met:

- 1) The side on the *away_layer* is within the specified number of sides away from the intersecting side on the other layer that is too close.
- 2) The angle between the sides is greater than the specified angle.

Let us consider the example shown in Figure 67. (This geometry is stored in the AWAY.CEL file supplied with the beta version update.) Let us assume that we need to find sides of layer B that are within 20 user units of sides of layer A shapes. However, we consider only parallel sides within this distance as true errors. We want to mark only sides 3 and 4. Perpendicular side pairs and sides at a 45° angle are permitted and should not be marked as errors.

The perpendicular sides 5 and 6 could be prevented from being marked as errors by adding the /~P option to the rule. You could prevent marking the crossing sides with the /~CROSS option. The layer B sides within the layer A shape can be prevented from being considered as errors by adding the /OUT option to the layer A specification in the MIN_SPACING rule.

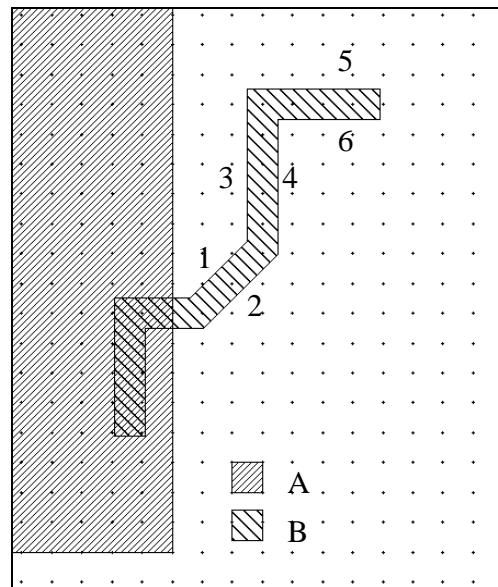


Figure 67: AWAY.CEL

Without the AWAY option, there is no way to prevent sides 1 and 2 from being marked as errors. However, when we use the AWAY option in the MIN_SPACING rule we can prevent the DRC from marking sides 1 and 2. Then the DRC will mark only sides 3 and 4 as errors.

Beta Test Warning

The new algorithms required to implement the AWAY option required changes to the MIN_SPACING algorithms. You should verify the results of all MIN_SPACING rules tested with this beta version. This includes the results of MIN_SPACING rules that do not use the AWAY option. Verify all MIN_SPACING results produced by this beta version against the results of the released version.

If the results of any MIN_SPACING rules are different between the versions, please contact IC Editors.

We do run a test suite comparing the new and old versions before we post a beta version. But just because our cases worked, that doesn't mean yours will.

End Caps

One more optional parameter applies only to the layer it follows. The /CAP=*angle* parameter is used to avoid marking sides that lie within the end cap of a side on the indicated layer. The angle must be in the range 90:180. The angle is measured from the edge.

Let us first consider the region checked for various values of the cap angle for a given single side. You can see from Figure 68 that when the cap angle is 180° , the entire region around the side is checked. This is the default when no /CAP keyword is used.

When the cap angle is set to 90° , only sides on the other layer that are within the shaded area shown will be considered errors.

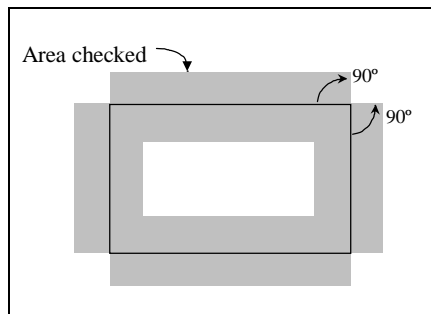


Figure 69: Region around a rectangle that is checked when /CAP=90 is used.

You can combine the /IN or /OUT specifications with a cap angle specification.

Example:

**ERR=MINSACING(A, ...
... B/OUT /CAP=90 , 1)**

When both the /OUT and /CAP=90 options are added to the layer B specification, the region checked will look like Figure 70.

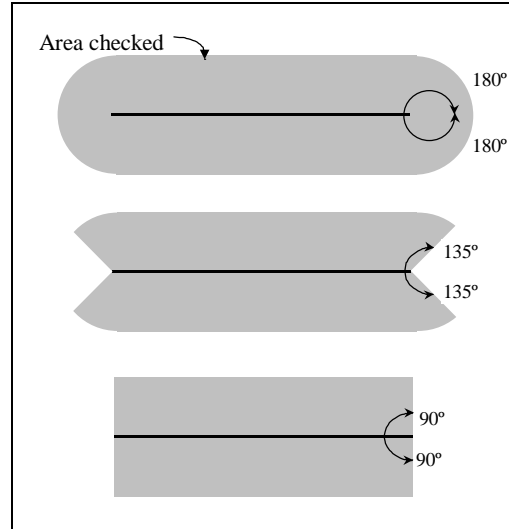


Figure 68: Region around a single side that is checked when various cap angles are set.

When you consider all sides of a given shape, remember that each side has an end cap. See Figure 69 for the region checked around a rectangle when the cap angle is set to 90° . Only sides of shapes on the other layer that are in this region will be marked as errors.

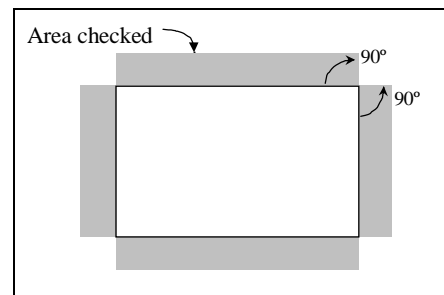


Figure 70: Region checked with /OUT and /CAP=90 options.

Orientation Options

These MIN_SPACING options can prevent side-side pairs from being considered errors based on the orientation of the sides with each other. These criteria are applied **after** the directional and end cap criteria are applied.

Each of the orientation options is set in every MIN_SPACING rule. Only when the option is preceded with a '~', will side-side pairs in that orientation be prevented from being considered errors. For simple spacing checks, the default is to consider all of these orientations as errors. However, directional spacing checks (any MINSPACING rules that use /IN or /OUT keywords) will by default **not** mark as errors crossing, t-intersection or perpendicular side-side relationships. If you consider one or more of these relationships to be errors in a directional spacing rule, you must override the defaults.

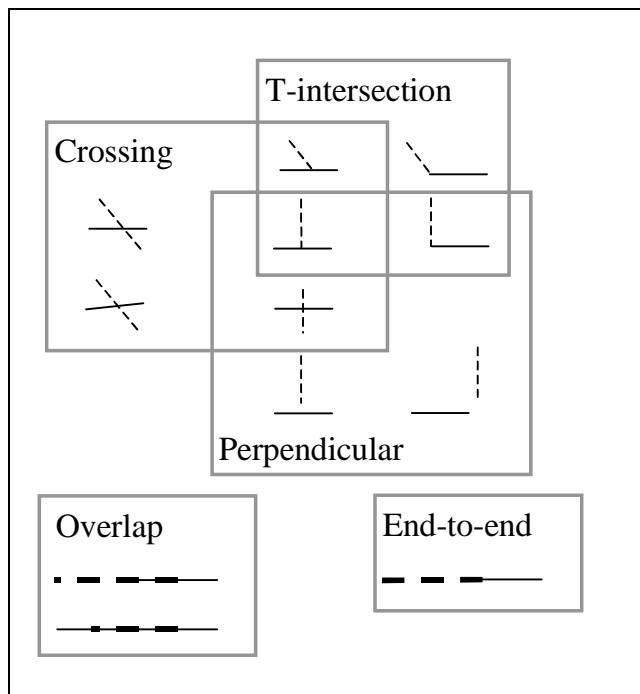


Figure 71: Various side-side orientation relationships.

Example: **ERR=MINSACING(A, B ,1.1 /~CROSS)**

When this rule is run on the shapes shown in Figure 72, violations between sides that cross will not be marked. Only the parallel sides closer than 1.1 units will be marked. If the /~CROSS option was not used the vertical sides of the cross-hatched wire would be marked where they cross the other wire.

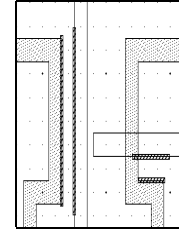


Figure 72:

One pair of sides may have more than one relationship. If one of these relationships is indicated with a '~' in the MIN_SPACING rule, that is enough for the pair to not be considered an error.

If you have difficulty determining why a side is marked, turn on detailed logging. See page 50.

A side that is eliminated as an error from one pair due to the orientation options may still be in error with a different side.

The exact definitions of the various orientations are:

Many examples of these types of intersections are provided in the syntax description of the MIN_SPACING rule. See page 252.

Crossing intersections: Intersections where the sides share a single point and at least one side continues on both sides of the point of intersection. The sides cannot meet at 0° or 180°; i.e. the sides cannot overlap or meet end-to-end.

T-intersections: The sides must share a single point and that must be the end point of one of the sides. The sides cannot overlap or meet end-to-end.

Perpendicular relationships: The sides must be exactly 90° from each other. The sides do not need to intersect.

Overlaps: The sides must share a non-zero length.

End-to-end intersections: The end points of the sides meet and the sides are at 180° from each other.

Electrical Connection Criteria

If you define how electrical connections between layers are formed in your layout, the DRC can tell which shapes are electrically connected. When you add the /CONN option to a MINSPACING rule, only shapes that are electrically connected will be considered as potential errors. When you add the /~CONN option, only shapes that are not electrically connected will be considered potential errors.

The default DRC behavior is to check spacing between both electrically connected and unconnected shapes.

If electrical connections are a criteria in your spacing check, you need to define electrical connections with CONNECT rules.

The QUICK_PASS command line parameter **must not** be used when you want electrical connection criteria to be applied. If you do use the QUICK_PASS algorithms, the electrical connection criteria are ignored without warning.

Error Wire Length Criteria

Some spacing rules allow two shapes to be closer than the minimum distance if the length of the sides in violation is shorter than a minimum length. When short spacing errors are false errors, you can use the /LENGTH option of the MIN_SPACING rule to discard error wires that are shorter than a specified length. The discarded error wires are not added to the error count.

Using this feature can result in unpaired error wires.

Example:

```
ERR=MIN_SPACING ...  
...(A, A, 2 /LENGTH=4)
```

When this rule is run on the shapes in Figure 73, the two boxes on the top are in violation with the long box

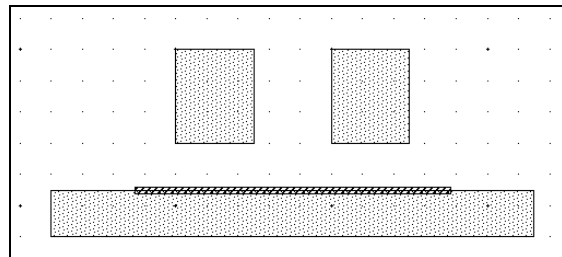


Figure 73: Unpaired error wire.

How the DRC Works: Spacing Verification

below. However, the error wires that indicate which sides are in violation for the two top boxes are shorter than 4 units. They are discarded by the DRC. The longer error wire on the long box is kept.

When it is difficult to determine why a single side is marked when you have used length criteria, you can turn on detailed logging to list each pair of sides in error. Errors that are discarded due to length criteria are still listed in the messages in the log file when detailed logging is enabled.

When you use error length criteria in any MINSPACING rule, the DRC will automatically invoke the slower spacing check algorithms. If you override this default by using the QUICK_SPACING command line option, you may prevent errors from being found. We cover this subject next.

QUICK_SPACING Algorithm

The DRC can use one of two different algorithms for verifying MIN_SPACING rules. The DRC will automatically choose the algorithm based on the contents of your rule set and whether or not the entire design is being checked.

The faster algorithm will be chosen automatically if it can not cause errors to be missed. Using this algorithm can reduce processing time on the order of 10%.

When the faster algorithm may cause errors to be missed, the DRC will automatically use the slower algorithm. However, you can override this behavior by adding the QUICK_SPACING keyword to the DRC command line.

The type of error that may be missed involves vertices of shapes that occur outside of the area being checked. Look at Figure 74. Let us say that these two shapes violate a MINSPACING rule. However, since the vertices lie outside of the area checked, the quicker spacing algorithm will miss the error.

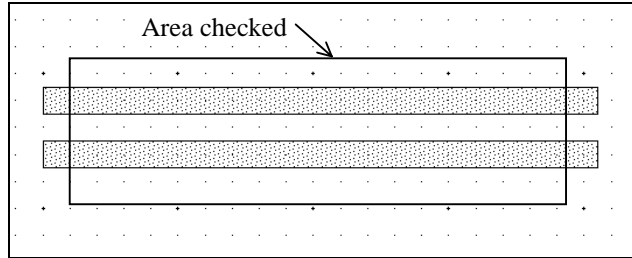


Figure 74: Shapes with vertices outside area checked.

There are two situations where the faster spacing algorithm may miss errors:

The design area verified is limited by the LEFT, RIGHT, TOP, or BOTTOM keywords on the DRC command line.

The area checked is the current panel, including the border. The error would be caught in the panels to the left and right. However, if the /LENGTH option is used in the MIN_SPACING rule, the errors may be missed due to being too short in the panels on the left and right.

If you will be performing several DRC runs on your design, you can add the QUICK_SPACING option to some runs to save time. However, your final runs should not use this option or errors like the one above may be missed.

How the DRC Works: Spacing Verification

Subject	Importance	Page
MIN_SPACING rule	DRC rule to verify distance between different shapes	252
Example of separating wires by size	Sample method to follow when you need to apply different MIN_SPACING rules to irregular shapes on a layer based on size.	65
Electrical Connections	Overview of defining electrical connections to restrict spacing errors	110
Advance tutorial examples	Simple spacing example	381
	Directional spacing examples	391
	Using TOUCHING test for enclosure	397
	Electrical criteria example	402
CONNECT rule	Syntax of rule to define electrical connections	200
QUICK_PASS command line option	Causes the DRC to ignore electrical connection criteria in a MIN_SPACING rule	129 and 337
QUICK_SPACING command line option	Can cause the DRC to miss MIN_SPACING errors in rare cases	338
Command file description	Hints on using layout editor commands to make error wires easier to see	365
Detailed logging	Add coordinates of specific pairs of sides marked by MIN_SPACING rules to log file	50

Figure 75: References for spacing verification

Other Verification Rules

These rules automatically add all shapes created to the error count in the same manner as the MIN_SPACING rule.

Width and Notch Verification

The MIN_WIDTH and MIN_NOTCH rules are very similar rules to test minimum distances within single shapes. MIN_WIDTH verifies the distance between all non-adjacent sides of a given shape where the distance is measured through the material. The MIN_NOTCH rule verifies the distance between all non-adjacent sides of a given shape where the distance is measured through a gap in the material.

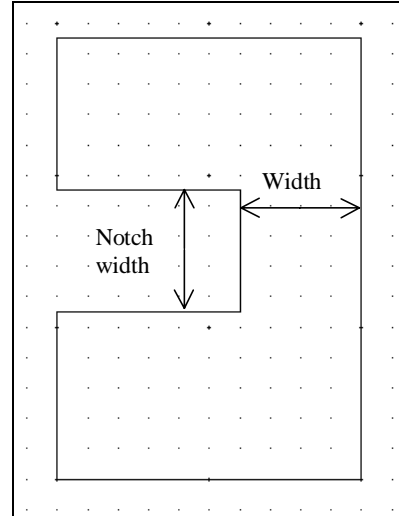


Figure 76: Example of width and notch measurement

DRC Definition of Width

The exact DRC definition of the width of polygons is surprisingly complex. It helps to visualize tracing the sides of a polygon like the one in Figure 77. Begin at one vertex of polygon, trace clockwise around each side indicating the sense of direction of each edge. The DRC will search for edges in violation of the width restriction only to the right of each edge, toward the interior of the polygon.

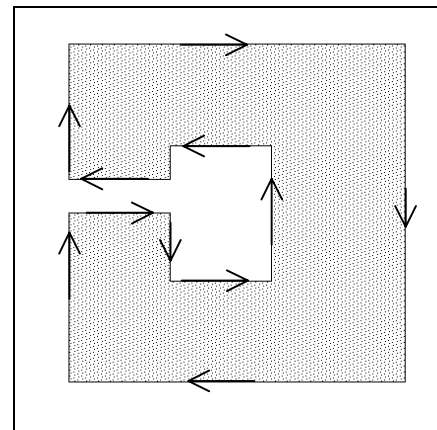


Figure 77: Edges of polygon with directions marked.

How the DRC Works: Other Verification Rules

Edge A has a width violation to edge B when all of following conditions are met:

- 1) The edges are closer than the distance specified in the MIN_WIDTH rule. (Sides exactly this distance apart are **not** marked.)
- 2) The edges do not share a vertex (i.e. adjacent sides cannot have a width violation.)
- 3) Edge B is located toward the interior of the polygon from edge A. Remember that the DRC will look only to the right of the edge in question.
- 4) The angle between the sense-of-direction vectors for each edge must be greater than 90° .

Let us cover some examples that demonstrate conditions 2 through 4. (Number 1 is fairly easy to understand.)

Condition number 2 can be illustrated with angular protrusions. Look at the polygon in Figure 78. The protrusion on the left has a bisecting side, and the protrusion on the right is formed from adjacent sides. The protrusion on the left will be marked by the MIN_WIDTH rule if these sides are closer than the distance specified. However, the protrusion on the right will not be marked by the MIN_WIDTH rule.

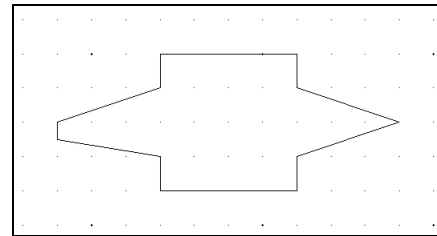


Figure 78: Polygon with angular protrusions.

The MIN_ANGLE or WARN_ACUTE rules can be used to find protrusions like the one on the right.

The reason for this condition is that if the program tested distance between adjacent sides, every corner of every polygon could be marked as an error. This would definitely be undesirable.

Our next example demonstrates some of the ideas behind conditions 3 and 4. Side 1 in Figure 79 may see side 2 as a violation since it is located toward the interior of side 1. However, it will not be marked due to condition 4. These sides are at 0° from each other.

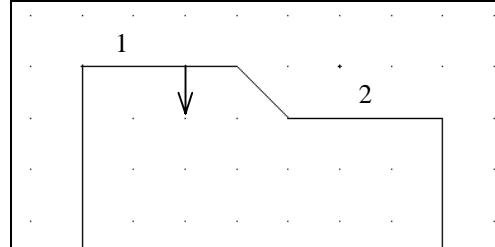


Figure 79: Polygon with non-adjacent sides at 0° .

Condition 4 will also prevent width violations from being marked for cut-off perpendicular corners. Most users would agree that the cut-off 90° corner of the shape on the left does not represent a minimum width error. However, it passes conditions 1, 2, and 3 of the width test. It is condition number 4 that prevents it from being a width violation.

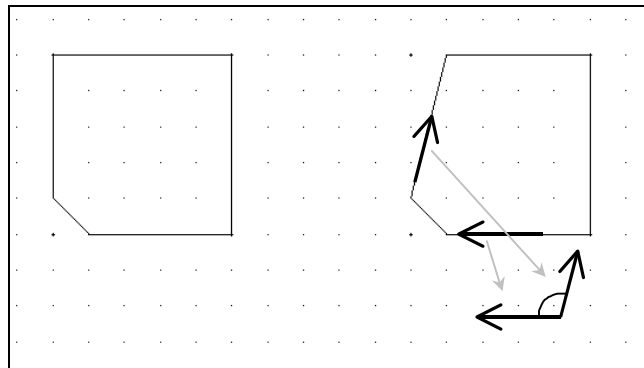


Figure 80: Two polygons with cut-off corners. Only the one on the right will be marked as a width violation since the angle between the sides is greater than 90° .

The shape on the right is a different story. This type of cut-off corner does represent a width violation. If you have forgotten how to measure the angle between vectors, remember that you relocate the vectors to align their bases before you measure the angle. When you do this, you can see that the angle between these sides is greater than 90° .

DRC Definition of Notch

The DRC definition of a notch is identical to the definition of width, except for condition 3. Now the DRC searches only to the left of each edge, toward the exterior of the polygon.

Angular notches have the same restriction as angular protrusions. If they are formed from adjacent sides, they will not be marked by the MIN_NOTCH rule. However, if there is a bisecting side like the notch on the left of Figure 81, the notch will be verified by the MIN_NOTCH rule.

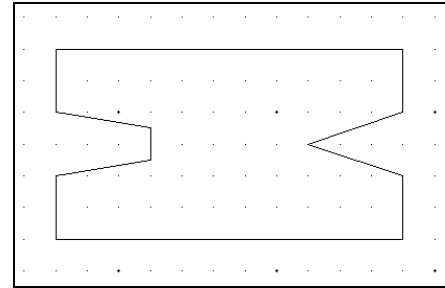


Figure 81: Two notches. Only the one on the left will be found by MIN_NOTCH rule.

The MAX_ANGLE or WARN_ACUTE rules can be used to find angular notches like the one on the right. Find holes with the ISLANDS or HOLE_AREA_FRACTION rules.

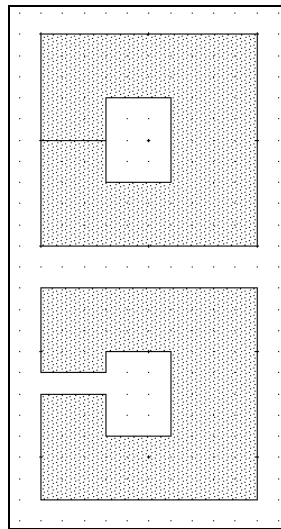


Figure 82: One enclosed hole and one open-ended hole with notch.

Our next example deals

with holes. Look at Figure 82. The shape on the top has an enclosed hole. The ICED™ layout editor cannot represent a shape with a hole unless the polygon has sides that connect the inside hole to the outside edges. You may think that the DRC will consider this a notch with a distance of 0 and mark it as an error. However, the internal representation in the DRC does not use such construction lines to connect holes to the outer boundary. The DRC layer preprocessing removes the unnecessary sides of the top shape that connect the inside edges with the outside edges. The DRC will test the top shape as an unseamed shape with a hole so the MIN_NOTCH rule will not mark it as an error. The shape in the bottom does have a notch that will be verified by the MIN_NOTCH rule.

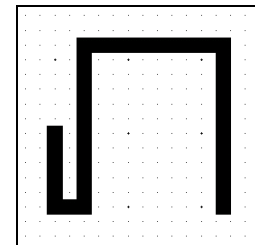


Figure 83: Notch, not MINSPACING error.

See page 130 for an example of a notch that will not be found when the QUICK_PASS option is used on the DRC command line.

Angular Notches and Protrusions

The MIN_ANGLE and MAX_ANGLE rules are used to find acute angles on shapes on specific layers. The MIN_ANGLE rule will find angular protrusions, while MAX_ANGLE will find angular notches.

Both rules measure the angle in the **interior** of the polygon. The MIN_ANGLE rule will mark all angles less than the angle specified in the rule. Use a value less than 90° to find only acute angular protrusions.

The MAX_ANGLE rule will mark all angles greater than the specified angle. Find acute angular notches by using a value greater than 270° but less than 360° .

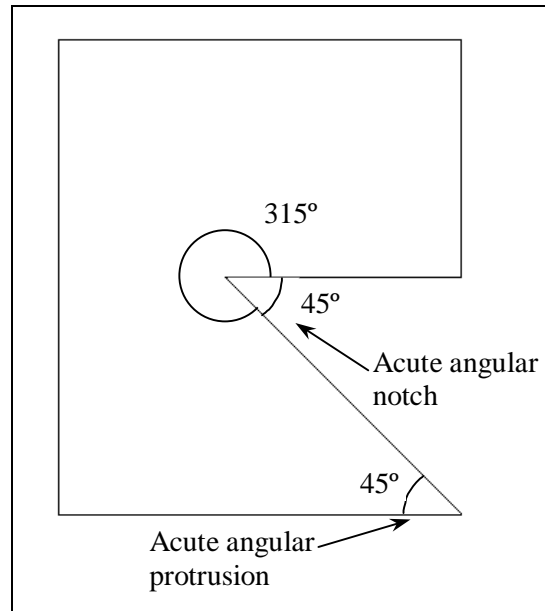


Figure 84: Acute angular protrusion and notch.

Minimum Area and Side Length

MIN_AREA

Some design rules define a minimum area for shapes on a given layer in addition to (or instead of) minimum width restrictions. This can be verified in the DRC with the MIN_AREA rule.

The MIN_AREA rule has a required parameter to determine how false errors due to panel boundaries will be prevented. If several touching shapes travel across panel boundaries, only the shapes in the current panel and neighbor panels are merged. If a shape is marked as an error by the MIN_AREA rule only

because pieces of it are not merged due to panel boundaries, this is a false error. See the syntax of the MIN_AREA rule (page 243) and the discussion of panel borders (page 124) for examples of how to avoid these false errors.

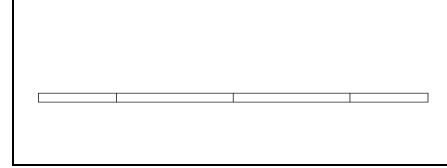


Figure 85: Touching shapes that form long wire.

While this rule is usually used as a verification rule where the shapes on the result layer are output to mark errors and added to the error count, you can instead use it as a filter to find small shapes without marking them as errors. See the example on page 66.

The table on page 62 lists the geometric basis and automatic error status of shapes created by each rule.

The MIN_AREA rule can be used as a filter because it generates shapes with a polygon geometric basis. The DRC can use these shapes in other rules in the same manner as any other rule that generates polygons. The shapes created by rules that generate wires cannot be used this way. Layers that contain wire shapes cannot be used on the right side of the '=' in any other rule.

Remember that the DRC contains other rules to classify shapes by size. The BOUNDS and IS_BOX rules both filter small shapes. If you want shapes generated by these rules to be counted as errors you must define the result layers with the ERROR keyword.

MIN_SIDE

The MIN_SIDE rule will mark all sides of polygons on the indicated layer that are less than the indicated length.

The MIN_SIDE rule **cannot** be used as a filter for isolating shapes that are verified by other rules. It generates wire shapes to mark sides. The result-layer generated by the MIN_SIDE rule cannot be used by any other rule. Shapes on the result-layer will automatically be added to the error count.

Design Area Coverage by a Layer

You may need to verify that a layer covers at least a certain minimum fraction of your total design area. The DRC rule MIN_FILL performs this function.

Subject	Importance	Page
MAX_ANGLE rule	Used to find acute angle notches	231
MIN_ANGLE rule	Used to find acute angle protrusions	242
MIN_AREA rule	Copies to an error layer all shapes with less than the indicated area	243
MIN_FILL rule	Verifies that a layer covers at least a minimum fraction of the total design area.	211
MIN_NOTCH rule	Marks sides too close to each other across a gap on a given polygon	248
MIN_SIDE rule	Marks short sides as errors	251
MIN_WIDTH rule	Marks sides too close to each other in a single polygon	271

Figure 86: References for other verification rules.

Electrical Connections

The QUICK_PASS option on the DRC command line will prevent electrical connections from being recognized.

If your design rules specify minimum spacing rules that depend on whether or not two shapes are electrically connected, the DRC can determine this. You must define how the electrical connections are formed before the MINSPACING rules that use this information. Then the /CONN or /~CONN options in the MIN_SPACING rule will use this information to eliminate false errors of shapes that are or are not electrically connected.

If electrical connections are not important to your spacing rules, you may want to skip this entire section. However, the information on the STAMP rule may be important if you want to verify that shapes like transistor wells are electrically connected to exactly one node.

The CONNECT and STAMP Rules

See an example of this process in the Advanced Tutorial on page 402.

Your rule set defines electrical connections using a combination of layer processing rules, the CONNECT rule, and the STAMP rule. Layer processing rules are used to create the conductive layers from the layers used in the layout and to remove device area from the conductive layers. The CONNECT rule defines which layers form electrical connections when they touch. The STAMP rule defines layers that are poor conductors.

A collection of shapes that are electrically connected to each other is called a **net**. The DRC recognizes which shapes are in the same net by assigning node numbers. Initially, each polygon in the DRC database is assigned a unique node number. As new polygons are formed on new generated layers, they are assigned new node numbers. When the CONNECT rules are processed, two touching polygons on electrically connected layers will both be assigned the lower node number of the pair. Eventually, all polygons in a net are assigned the same node number.

The STAMP rule is used primarily to insure that poor conductor shapes are connected to exactly one electrical net, rather than to define electrical connections for the MIN_SPACING rule. The STAMP rule assigns node numbers in a slightly different way. The STAMP rule will assign a poor conductor shape the same node number as the first touching conductive shape.

However, other conductive shapes that touch the poor conductor will not be assigned the same node number. We will cover the use of the STAMP rule later.

The simplest form of the CONNECT rule is:

CONNECT *layer1 layer2*

When a shape on layer1 touches a shape on layer2, both of them will be assigned the same node number. Both shapes will be considered part of the same net.

When shapes on two layers are connected by a shape on a third layer (e.g. a via or contact hole shape), you use this form of the CONNECT rule:

CONNECT *layer1 layer2 BY layer3*

When this form of the CONNECT rule is used, shapes on all three layers must share a common area for them to be electrically connected.

Example:

**CONNECT M1 M1WIRE
CONNECT M1 M2 BY VIA**

When the rules above are run on the shapes shown in Figure 87, all of the shapes will be electrically connected.

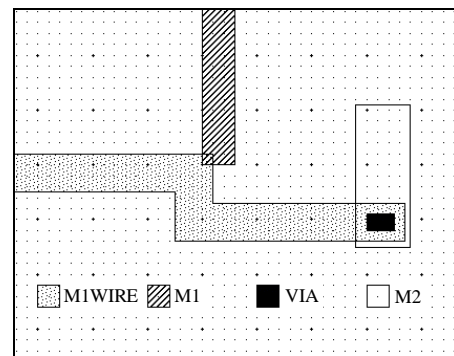


Figure 87: Electrically connected shapes.

Layers that are used in CONNECT and STAMP rules form **groups**. All layers that can be connected to each other are collected into a single group. When layers form more than one group, there is no way to electrically connect a shape on a layer in one group to a shape on a layer in separate group. This may point out mistakes in the rule set.

The number of groups is reported in the log file created by the rules compiler. If you have more than one group, you should look carefully at the log file where the layers in each group are listed to be sure that you are not forgetting to connect some layers. However, some temporary layers may form separate groups.

Building Electrical Connections

It can be trickier than you might think to build electrical connections correctly. It is very easy to short layers together unintentionally. When you write the rules for electrical connections, you must consider how a chip is fabricated. Keep in mind which layers prevent shorts, including which combinations of layers represent devices that break shapes into separate nets.

Example:

```
GATE =      DIFF AND      POLY
SRC_DRN =  DIFF AND NOT  POLY
CONNECT  M1 POLY      BY CONT
CONNECT  M1 SRC_DRN  BY CONT
```

You can use the NLE program to test your electrical connection rules. The node outliner utility will highlight entire electric nets. Refer to the NLE/LVS manual for more examples.

In this example of a MOSFET technology, the DIFF (diffusion) layer is separated into the GATE layer and the SRC_DRN (or source-drain) layer. The GATE layer is the device layer. The SRC_DRN layer represents the terminals of the device. We do not want to short the terminals of the device into one net.

Note that the DIFF layer is not used as a conductive layer in the CONNECT rules. Instead, the SRC_DRN layer is used as the conductive layer. If you used DIFF in the CONNECT rules, the source and drain of each FET device would be shorted together.

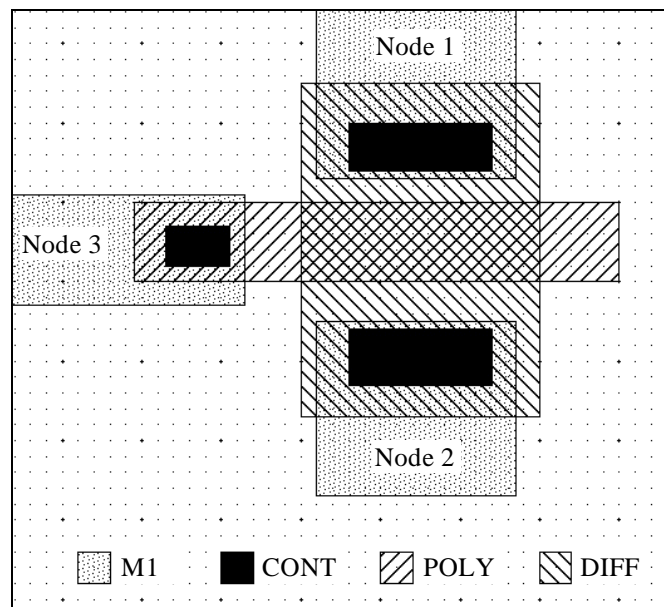


Figure 88: FET device.

There are four methods for removing material from conductive layers to avoid shorting the terminals of devices:

- 1) Use the AND rule to remove the device area from the conductive layer. This is the method used in the example above.
- 2) Shapes on a dummy layer are added to the design. This layer can then be used to etch the conductive layer using the AND rule, or the TOUCHING rule can be used to find shapes that touch the dummy layer and these are removed from the conductive layer. This method is often used to remove the area that represents a resistor from the conductive layer.
- 3) The IN_CELL rule (or the IN_CELL keyword of the INPUT LAYER rule) changes all shapes on a conductive layer contained in certain cells to a different layer. In this case, shapes in the main cell which travel over the same area will remain on the conductive layer.
- 4) IN_CELL processing is used to save layer 0 (which represents the bounding box of a cell) in certain cells to a scratch layer, which is then used to remove area from the conductive layer. In this case, shapes in the main cell which travel over the same area will also be removed from the conductive layer.

The NLE uses these same methods to recognize device areas. See the examples in the NLE/LVS manual if you will be using the same dummy shapes for device recognition using the NLE.

Example:

```
RESISTOR = POLY AND      RESMASK
POLY =     POLY AND NOT RESMASK
CONNECT  M1 POLY      BY CONT
```

These rules represent an example of method 2. Look at Figure 89. A shape has been added on the dummy layer RESMASK. This shape is then used to remove the area of the resistor from the POLY layer before it is used in the CONNECT rule.

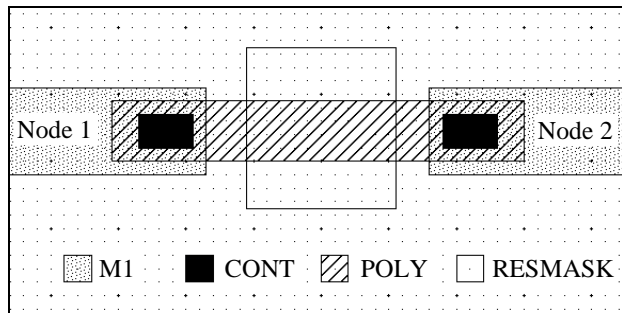


Figure 89: Resistor device.

How the DRC Works: Electrical Connections

See page 155 for an example of what happens when the layout changes leaving the shape on the dummy layer in the wrong place.

If you will not be performing verification rules on the RESISTOR layer, the size of the RESMASK shapes is not important. In this case, all that is important is that they cut each POLY shape into two nodes. However, if you have design rules to verify for the RESISTOR shapes, then add the RESMASK shapes carefully to accurately create the RESISTOR shapes.

Contact layers can also require careful handling before using them in CONNECT rules. The order in which the layers are laid down should be considered as you build the electrical connections.

Refer to Figure 90 and Figure 91 as we discuss the following example. The layer generation and connection rules for NPN transistors demonstrate how you must be careful not to short layers. (We have left out the buried layer to simplify the discussion.)

The P layer in a NPN transistor prevents the N_PLUS layer from contacting the N layer. Also, the N_PLUS layer prevents contacts from connecting M1 to the P layer. In this case, the contacts must be filtered to prevent several different layers from being shorted together.

Example:

```
N_AND_N_PLUS = N          AND      N_PLUS
EMITTER =      N_AND_N_PLUS AND      P
COLLECTOR =    N_AND_N_PLUS AND NOT P
BASE =         P
CONT_TO_BASE = CONTACTS    AND NOT  EMITTER

CONNECT      COLLECTOR N
CONNECT      M1 COLLECTOR BY CONTACTS
CONNECT      M1 EMITTER  BY CONTACTS
CONNECT      M1 BASE     BY  CONT_TO_BASE
```

If you created the COLLECTOR layer as follows:

COLLECTOR = N AND N_PLUS

then the shapes that make the emitter will also wind up on the COLLECTOR layer. In this case, the N layer will short the collector and the emitter. You must be careful to separate the COLLECTOR layer from the EMITTER layer by using the P layer.

You must classify the contacts that are over the P or BASE layer because the emitter is also over the P layer. Contacts over the emitter do not connect to the P layer, since the N_PLUS layer is in between. If the BASE layer is connected to M1 by CONTACTS, the emitter contact will short to the base since the emitter is on top of the BASE layer.

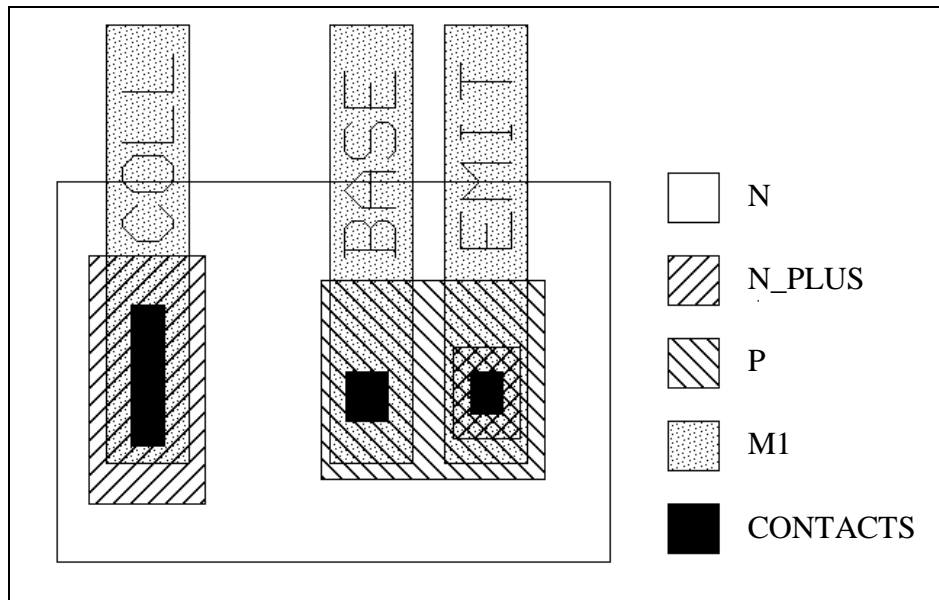


Figure 90: Simplified layout for a NPN device.

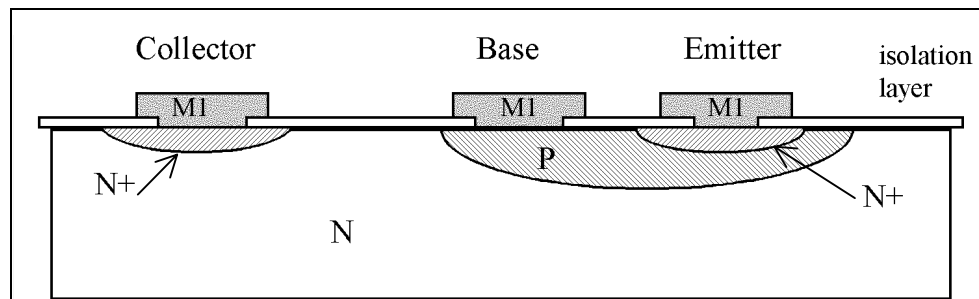


Figure 91: Simplified cross section of a NPN transistor.

Using the STAMP Rule to Verify Wells

We can demonstrate the importance of using the STAMP rule to verify transistor well or bulk layer shapes with Figure 92. Let us assume that the GND wire on the right connects to the metal GND bus and from there to a pad on the chip. However, the GND wire on the left does not connect to the bus. You meant to connect these two wires, but a gap exists by accident.

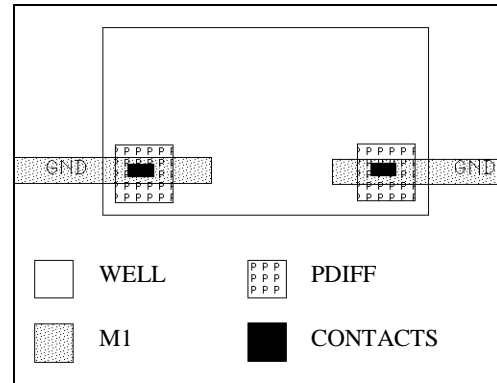


Figure 92: Open on GND node that connects only through WELL layer.

In this case, as long as you do not use the CONNECT rule to define electrical connections to the WELL layer, the two GND fragments will have different node numbers and you can find this error with the STAMP rule.

Example:

INPUT LAYER 1 NDIFF; 2 POLY; 3 WELL; 4 PDIFF;
INPUT LAYER 10 M1; 8 CONTACTS;
SCRATCH LAYER GATE; SRC_DRN;
OUTPUT LAYER 101 MULTI_WELL; 102 UNCONN_WELL

GATE = NDIFF AND POLY
SRC_DRN = NDIFF AND NOT POLY

CONNECT M1 PDIFF BY CONTACTS
CONNECT M1 SRC_DRN BY CONTACTS

STAMP WELL BY PDIFF MULTI=MULTI_WELL NONE=UNCONN_WELL

If this set of rules is run on the layout shown in Figure 92, the WELL shape will be copied to MULTI_WELL since it will be stamped by two different nodes on layer PDIFF. The two GND net fragments will not be shorted together and will be recognized by the DRC as two separate nets. All WELL shapes that do not have connections to PDIFF shapes will be copied to layer UNCONN_WELL. Shapes on both UNCONN_WELL and MULTI_WELL are automatically added to the error count.

Subject	Importance	Page
CONNECT rule	Used to define most electrical connections	200
STAMP rule	Used to define electrical connections for poor conductors and verify these connections	308
Advanced Tutorial	Example of set of CONNECT and STAMP rules for MOSFET process	402

Figure 93: References for electrical connection definition.

Panel Processing

Purpose

You can now specify panel size on the DRC command line. See the new PANEL options in Running the DRC.

The DRC is designed to verify large amounts of data. However, a whole chip can result in a huge database. At best, a large design can result in a huge scratch file and very long run times. At worst, the data will not fit on a typical disk drive. The DRC solves this problem by processing large designs in small panels.

Part of the problem is that verification rules must be executed on the design after it has been flattened hierarchically. Since the DRC does not require extra design constraints that prevent cells from overlapping each other, the only way to find violations that result from abutting or overlapping cells is to flatten the data so that these problems can be found.

The optimized algorithms require more data to be stored for each shape than a single set of coordinate data. Once you flatten the data, a typical chip probably cannot be stored as one flat unit.

The DRC can process small designs as a unit; however, larger designs may need to be automatically divided into panels and processed one panel at a time. Panel processing allows data that is not included in the current panel to be stored hierarchically. Only one panel is flattened at a time.

Unless you specify a maximum panel size, the DRC will attempt to find a suitable panel size based on the size and density of your design and the amount of memory available. However, if the DRC runs out of memory with this panel size, it will begin all processing again after dividing the design into panels half the size of the entire design. This process may be repeated with smaller and smaller panels. This type of thrashing may waste considerable time.

<p>If the DRC crashes due to memory problems, or with an error message that mentions panel size, read the following information carefully BEFORE calling technical support.</p>
--

Effect of Panel Size on Memory and Running Time

Even if the DRC can process your design in a single panel, the memory requirements may force the DRC to swap data to a scratch file on your hard drive. This is called disk swapping and it will result in long run times.

If you have a small amount of memory on your computer (less than 16Meg), then dividing your design into panels may allow the DRC to run to completion when it has run out of memory trying to process the entire design as one panel. Even if you have a large amount of memory on your computer, dividing the design into panels may speed up the DRC run by over an order of magnitude.

For example, a chip that took over 8 hours to process as a single panel took only an hour and a half to process when divided into "reasonable" panels. When you have long run times, you should divide the design into smaller panels.

One indication that the DRC will run faster if you specify smaller panels is when the log file from your first run reports that the DRC is swapping data to disk. The DRC reports at the end of the log file the size of the scratch file, the number of times it was used, and the percentage of processing time spent on swapping. If these numbers are large, trying a smaller panel size will probably result in a shorter running time. (In the testcase mentioned above, the log stated that 81% of the 8 hours was spent on disk swapping.)

See an example of the process of optimizing panel size on page 445.

If you will be running the DRC many times on your design, you should experiment with different panel sizes to find an optimum panel size. This can speed up the DRC processing time dramatically.

During development of the new panel size defaults, some testcases had the fastest run time with the default panel size. However in one testcase, the default panel size resulted in a run that took 4 times as long as a run with an optimized panel size.

The PANELX and PANLEY rules explicitly set panel size.

If your design is processed with panels that are just a little too large based on the amount of memory available, the DRC may run out of memory and try to recover by reprocessing all data from the beginning with panels half the size. Hours of processing time may be wasted. In this case the console messages look similar to the following:

How the DRC Works: Panel Processing

```
Panel 4, from x=570.167 to 1236.83, y=6570.58 to 7237.29 was too complex
on rule 98, pass 3.
Failure 14, 14 Checking spacing, too many vertices
Panel dimensions were 666.667 by 666.708
Try subdividing panel.
New panel dimensions are 666.667 by 333.354
Can only allocate 3559 size 82 items for processp--10, available=291887
Requested 28180 size 82 items=2310760 bytes
```

```
-----
Panel 6, from x=570.167 to 1236.83, y=6570.58 to 6903.94 was too complex
on rule 118, pass 3.
Failure 402, processp--10
Panel dimensions were 666.667 by 333.354
Try subdividing panel.
New panel dimensions are 333.333 by 333.354
Can only allocate 3559 size 82 items for processp--10, available=291887
Requested 28180 size 82 items=2310760 bytes
```

```
.      (many more similar messages)
.
```

```
Panel 25, from x=580.583 to 585.791, y=6575.79 to 6581 was too complex on
rule 118, pass 3.
Failure 402, processp--10
Panel dimensions were 5.20831 by 5.20869
Try subdividing panel.
Panel is too small to subdivide further.
```

Sometimes, data in large panels inherited from previous passes makes it impossible to subdivide panels in future passes. Try a smaller initial panel size.

Run aborted.

```
**CRASH*****CRASH*****CRASH*****CRASH*****CRASH**
**CRASH*****CRASH*****CRASH*****CRASH*****CRASH**
**CRASH*****CRASH*****CRASH*****CRASH*****CRASH**
```

If the console messages look similar to the ones above, try specifying panel sizes in the rule set about 10% to 25% of the first panel size reported. If this does not work, try even smaller panels in the rule set to avoid problems in future runs **before** you call technical support.

New Default Panel Size Calculations

As of version 3.14, the DRC attempts to calculate optimal panel size based on design size, density, and available memory. (Previous versions always defaulted to processing the data as a single panel unless the PANELX and/or PANELY

rules were used.) This automates the panel size selection process, and most designs may complete with acceptable run times with this default behavior.

The NO_PANELS rule forces the DRC to use a single panel the size of the design.

If your rule set was created for previous versions of the DRC, and includes PANELX and PANELY rules to explicitly set the panel size, you may see an improvement in run time by trying the new default panel calculations. Try removing the PANELX and PANELY rules for a test run if:

- The amount of memory available to the program has changed since the panel size was optimized. More memory may mean that larger panel sizes can now be used and may speed up processing.
- The density or size of your design has increased since you optimized the panel size. The larger database may be processed more efficiently with smaller panels.

However, the calculated default is not usually optimal, and large designs will almost certainly execute more quickly when an explicit maximum panel size is specified in the rule set.

If you do not specify the panel size, and the DRC run takes more than a few minutes, the DRC will add a warning to the console messages and log file similar to the following:

```
*****WARNING*****WARNING*****WARNING*****WARNING***  
    You used the default panel size. This will  
provide a panel size that does allow your job to run,  
but is unlikely to be the size that yields the fastest  
running time.
```

In this case, look at the log file to see the panel size calculated by the DRC and test panel sizes larger and smaller to find an optimal size. You can refer to the section of the advanced tutorial that covers this process on page 445.

The New PANEL_VERTICES Rule

If the default panel sizes seem to not be optimal given your design and memory constraints, one option is to “tweak” the automatic panel calculations with the PANEL_VERTICES rule rather than resort to explicit panel sizes with the PANELX and PANELY rules.

DRC memory is divided between main memory and data storage memory. See details on page 161.

The PANEL_VERTICES rule controls the maximum number of vertices in a panel, rather than the exact size of a panel. The value is specified as the number of vertices per panel per Megabyte of main memory available to the DRC, or:

$$\frac{\text{\# Vertices}}{\text{\# Panels} * \text{Megabytes Main Memory}} = \text{PANEL_VERTICES}$$

By default, PANEL_VERTICES is set to 5000. This provides an optimum number of vertices in a panel for some designs. If you have 50 Megabytes of main memory available to the DRC, this results in the following equation:

$$\frac{\text{Max \# Vertices in a Panel}}{50} = 5000$$

or

$$\text{Max \# Vertices in a Panel} = 250,000$$

If the total number of vertices in your design was 25 million, then the design would be divided equally into at least 100 panels. The DRC will test each panel to insure that the number of vertices on relevant input layers never exceeds 5000, even in dense areas of the design.

Since there is a trade off between extra processing required for panel processing and time saved due the smaller amount of data stored in flattened form at any given time, time may be saved by increasing or decreasing the default number of panels.

- If a run with the default number of panels completes successfully, you can see if a different number of panels leads to faster run times by specifying different PANEL_VERTICES values. The DRC log file lists the amount of time spent by each phase of the processing near the bottom of the file. If the

log file indicates that the DRC is spending significant time swapping data to disk, try adding a PANEL_VERTICALS rule in your rule set with a number smaller than 5000. If the log file indicates that little or no time is spent swapping data to disk, try increasing the panel size by with a PANEL_VERTICALS rule using a value larger than 5000.

- **On the other hand if the DRC crashes with a message that indicates a memory or panel size problem, or if disk swaps are slowing your run, try a number smaller than 5000 in the PANEL_VERTICALS rule.**

You can significantly decrease the amount of time the DRC takes to complete a run by optimizing panel processing. Try various values for PANEL_VERTICALS until you come up with an optimal value for your computer and design. Alternately, you can specify the panel size directly with the following rules.

The PANELX and PANELY Rules

The PANELX and PANELY rules are used to explicitly set the maximum panel size. If the PANEL_VERTICALS rule does not seem to provide you with a panel size that is working, you can use these rules to specify panel size.

You can
override the
panel size on
the DRC
command line.

The DRC attempts to divide the design into roughly equal panels. The dimensions you specify with the PANELX and PANELY rules are really the maximums rather than the exact dimensions used. If you specify PANELX = 100 and PANELY = 200 and your chip is 190 by 489 units, the chip will be divided into six 95 by 163 unit panels.

The optimum panel size varies greatly depending on the size of your design, the dimensions of your shapes, and on the type of rules you are processing. However, a rough rule of thumb, if most of your shapes and rules involve dimensions on the order of a few ICED™ units, is to use panels on the order of 300 by 300 units. (If you have less than 32 Megabytes of memory in your computer, you may want to start with panels smaller than this.) Try various sizes in succeeding runs to see which values give you the fastest run times.

Panel Borders

When you slice a design into panels for verification, shapes near the edge of a panel must be considered. If you are verifying a MIN_SPACING rule and a shape just inside the edge of one panel is too close to a shape in a neighboring panel, will the error be found? If panels did not overlap, many rules would miss errors when nearby shapes are not considered.

In order for shapes near or crossing a panel border to be processed correctly, the DRC must include a border around all sides of each panel. Shapes in the border area will be processed at least twice (at least four times near the corners of panels). Very small panels or very large borders will result in some shapes being processed many times. However, borders that are too small may allow errors to go undetected.

Add the
SHOW-
_BORDER
option to the
DRC command
line to see how
the border is
calculated by
the DRC.

The panel border is automatically calculated by the DRC based on the layer with the maximum reach as determined by the rules. **Reach** is defined as the minimum border distance that insures that no violations of shapes on a particular layer will be missed or marked as false errors.

Rules that involve touching have a reach of 0 due to the way the DRC processes touching shapes. (We cover this subject next.) Rules that involve dimensions and rules that involve changing the dimensions of shapes (like the BLOAT rule) require a reach to insure that shapes are processed correctly.

Each layer is initially assigned a reach of 0. Rules may increase this reach. The reach of a *result_layer* is often greater than the reach of the layers used to create it. The border is defined as the maximum reach of all layers plus a tiny safety factor.

The ASPECT_RATIO and MIN_AREA rules cannot compute reach, so you must specify it explicitly in the rule.

Rule	Reach of <i>result_layer</i>
AND	$\max(\text{Reach}(\text{layer1}), \text{Reach}(\text{layer2}))$
ASPECT_RATIO	$\text{Reach}(\text{layer1}) + \text{max_size parameter}$
Assignment Rule	$\text{Reach}(\text{layer1})$
BLOAT	$\text{Reach}(\text{layer1}) + \frac{\text{offset_val}}{\sin(\text{bloat_angle} / 2)}$
BOUNDS	$\text{Reach}(\text{layer1}) + \max(\text{size dimension})$ (if $\max(\text{size dimension}) > 10$, reach is 0 instead and second pass is added)
BRIDGE	0 (forces multiple passes)
CONNECT	0 (forces multiple passes)
IN_CELL	$\text{Reach}(\text{layer1})$
IS_BOX	$\text{Reach}(\text{layer1}) + \max(\text{size dimension})$
ISLANDS	0 (forces multiple passes)
MAX_ANGLE	$\text{Reach}(\text{layer1})$
MIN_ANGLE	$\text{Reach}(\text{layer1})$
MIN_AREA	$\text{Reach}(\text{layer1}) + \text{maxsize}$ (or 0 with extra pass if $\text{maxsize}=0$)
MIN_NOTCH	$\text{Reach}(\text{layer1}) + \text{min_width}$
MIN_SIDE	$\text{Reach}(\text{layer1}) + \text{min_length}$
MIN_SPACING	$\max(\text{Reach}(\text{layer1}), \text{Reach}(\text{layer2})) + \text{distance}$
MIN_WIDTH	$\text{Reach}(\text{layer1}) + \text{min_distance}$
OFF_GRID	$\text{Reach}(\text{layer1}) + \text{grid_resolution}$
OR	$\max(\text{Reach}(\text{layer1}), \text{Reach}(\text{layer2}))$
OVERLAPPING	0 (forces multiple passes)
SHRINK	$\text{Reach}(\text{layer1}) + \frac{\text{offset_val}}{\sin(\text{bloat_angle} / 2)}$
SNAP	$\text{Reach}(\text{layer1}) + \text{grid_resolution}$
SNAP45	$\text{Reach}(\text{layer1}) + \text{grid_resolution}$
STAMP	0 (forces multiple passes)
TOUCHING	0 (forces multiple passes)
XOR	$\max(\text{Reach}(\text{layer1}), \text{Reach}(\text{layer2}))$

Figure 94: Reach calculation for each rule.

How the DRC Works: Panel Processing

See the BLOAT-
_ANGLE rule
on page 311 for
more details on
bloating acute
angles.

The SHRINK and BLOAT rules can increase reach dramatically if you allow bloats of acute angles in your design.

For example, look at the small polygon with the 30° angle in Figure 95. When this polygon is bloated by 2 units without constraints, the bottom dimension expands from 10 to more than 20.

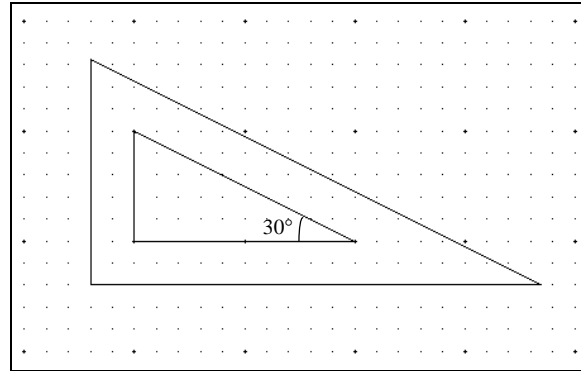


Figure 95: Unconstrained bloat of 30° angle.

If you do not
have acute
angles in your
design, you
should always
use the
maximum
BLOAT-
_ANGLE of 45°
(the default) to
avoid excessive
reaches.

The reach of a bloated layer is defined as:

$$\text{Reach}(\text{layer1}) + \text{offset_val} / \sin(\alpha / 2)$$

Where α is the *bloat_angle* parameter defined with the BLOAT_ANGLE rule. If the reach of layer1 is 0, the *offset_val* is 2, and the *bloat_angle* is 30, the reach of the bloated layer will be:

$$0 + 2 / \sin(30 / 2) = 7.27$$

The reach increases dramatically as the bloat angle decreases. If the bloat angle is set to 1, allowing unconstrained bloats of angles as small as 1°, the reach of the bloated layer in the example above goes up to 229.

The BOUNDS and IS_BOX rules add a reach to the *result_layer* equal to the amount of the maximum dimension you are verifying. If we use a BOUNDS rule with a maximum dimension of 10 units on a bloated layer with a reach of 7.27, the reach of the *result_layer* created by the BOUNDS rule is now 17.27.

If you use the layer created by the BOUNDS rule in other rules, the reach may go up even more. A reach this large is required to be absolutely sure that no polygons are improperly processed. However, this reach may be excessive for the other layers. Many polygons will be processed multiple times due to the large border.

The layer with the largest reach sets the border for all layers in a single pass. The border changes from pass to pass. Remember that a pass is defined as a collection of operations that can be performed with one sweep through all shapes in the database. Some passes require a large border due to the layers processed in that pass. Other passes require no border.

If your rule set is verifying minimum spacing rules on the order of 3 microns, this will result in a 3 micron reach for at least one pass. If your panel size is 100 microns, the DRC will have to perform duplicate processing on 12% of your design. This should not result in excessive run times.

See page 442 for an example of separating long reach rules from short reach rules.

However, if your rules require a reach of 25 microns (not unusual when testing pad design rules), then duplicate processing is performed on 50% of your data. This will include many rules that do not require such a large border. This will probably lead to excessive run times and memory usage. The best solution to this type of problem is to remove the long reach rules to a separate rule set that you run less often. You can increase the panel size for this smaller set of rules to further reduce duplicate processing.

See page 164 for a trick to reduce DRC running time by processing long reach rules as Boolean rules.

The panel border used by the DRC is reported in the log file. Search for the phrase “Panel Border”. You can use the `SHOW_BORDER` option on the DRC command line to report the reach and border calculations performed by the DRC. You can sometimes rewrite rules to reduce the reach.

If you are a DRC expert, and your rule set creates a border that you know is excessive; you can override the border calculated by the DRC with the `BORDER` rule or the `BORDER` option on the DRC command line. However, **if you don’t know exactly what you are doing, you can easily prevent real errors from being found by tampering with the border.**

If you have a large border and the DRC needs to reduce the panel size to run to completion, your run may be aborted with the message, “Panel is too small to subdivide further – check aborted”. This means that the border is at least one half the new panel size selected by the DRC. You will need to reduce the border by rewriting your rules or increase the memory available to the DRC so it can complete with larger panels. You can modify the border explicitly with the `BORDER` rule, but remember that you can corrupt the validity of the DRC tests by doing this.

Multiple Pass Processing

If a polygon crosses a panel plus border boundary, the DRC will use the whole polygon when processing each panel. A shape in one panel that touches a shape on the same layer in the border area will be merged with the other shape into a single polygon when that panel is processed. Touching shapes that are beyond this area will not be merged into single shapes for the current panel.

To avoid missing errors due to touching shapes outside panel borders, the DRC adds extra processing. All shapes in the database will be processed several times in multiple **passes** through the data. This extra processing can be turned off though the use of the **QUICK_PASS** option (which is covered on the next page).

Look at Figure 96. Let us assume that we will be verifying the three configurations with a **MIN_NOTCH** rule. The notch in the top configuration will be recognized since the shape remains a single shape despite the panel boundary. The DRC will see the notch in the middle configuration because the shapes will still be merged in either panel.

The configuration on the bottom will not be merged into a single shape. The automatic calculation to add a border around the panel cannot solve this problem because the shape that connects the two horizontal shapes may be a great distance away. The notch will be missed unless the DRC uses a different method to see that the horizontal shapes are really part of a single connected shape.

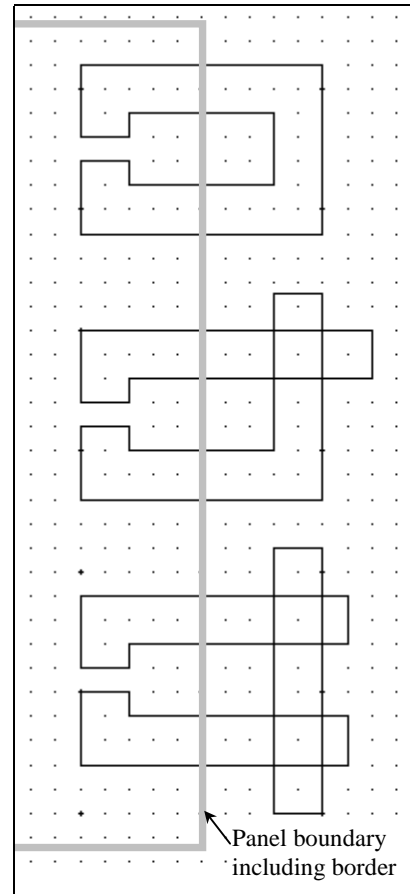


Figure 96: Panel processing will prevent DRC from merging the shapes in bottom configuration.

To avoid incorrectly processing touching shapes that cross panel boundaries, the DRC will automatically add special cases of the CONNECT rule to the processing. The CONNECT rule assigns unique polygon numbers to all shapes. Touching shapes have their numbers reassigned so that both shapes have the same polygon number. Once a CONNECT rule is processed, the DRC can recognize that two shapes are really part of the same connected shape because they have the same polygon number. In the example above, the DRC will see that the two horizontal shapes in the bottom configuration form a notch because the polygon numbers of both shapes will be the same.

These CONNECT rules add extra processing time. The rules that generate a layer must be executed in one pass through the database, then the CONNECT processing must take place in a separate pass. Finally, the rules that verify the shapes on that layer must be executed in a third pass. The extra time added to the DRC run is due not only to processing the CONNECT rules, but also to the storage of each panel of data for the next pass.

Effects of the QUICK_PASS Option

The number of passes required to execute the rules is listed near the bottom of the rules compiler log file.

If the DRC can execute your rules in a single pass through the data it will do just that. In this case, the DRC will not need to save the layout data from each panel for other passes. When your rule set includes rules that require the DRC to determine which shapes touch each other, the DRC must perform intermediate passes through the data to insure that all errors are found. In this case, the layout data in each panel must be saved for the next pass through the data.

When your rules require multiple passes, you must make a choice about how the DRC will proceed. Either the SLOW or QUICK_PASS option must be added to the command line. The SLOW option directs the DRC to process the layout data in multiple passes. This insures that all errors will be found. The QUICK_PASS option forces the DRC to ignore the problems caused by panel boundaries and touching shapes. The DRC will execute in a single pass. This may save a significant amount of processing time. However, some rules will not handle shapes crossing panel boundaries properly leading to false errors and even missed errors.

When you use the QUICK_PASS option on the DRC command line, you force the DRC to use faster algorithms that ignore some information relative to rules like minimum notch and width rules. This option is provided to allow for faster intermediate runs on layouts that are checked several times. One of the reasons for the speed improvement is that touching shapes outside the panel border will be ignored. In rare cases this can result in false errors or missed errors for the MIN_NOTCH and MIN_SPACING rules.

Let us consider the effect of the QUICKPASS option on the three notch configurations covered on page 128. Assume that the DRC will be verifying a MIN_NOTCH rule that all three configurations violate.

Even when the QUICK_PASS option is used, the DRC will recognize that the top two configurations each represent a single shape with a notch.

When the SLOW option is used instead of QUICK_PASS, the CONNECT rules force the DRC to recognize that the three shapes on the bottom are connected and so it will find the bottom notch.

When the QUICK_PASS option is used, the violation of the bottom configuration will not be found. The touching shape on the right will be ignored while the DRC processes the panel on the left. This means that the DRC will see the two other shapes as separate shapes on the same layer. The notch will not be found. A false MIN_SPACING error may be generated.

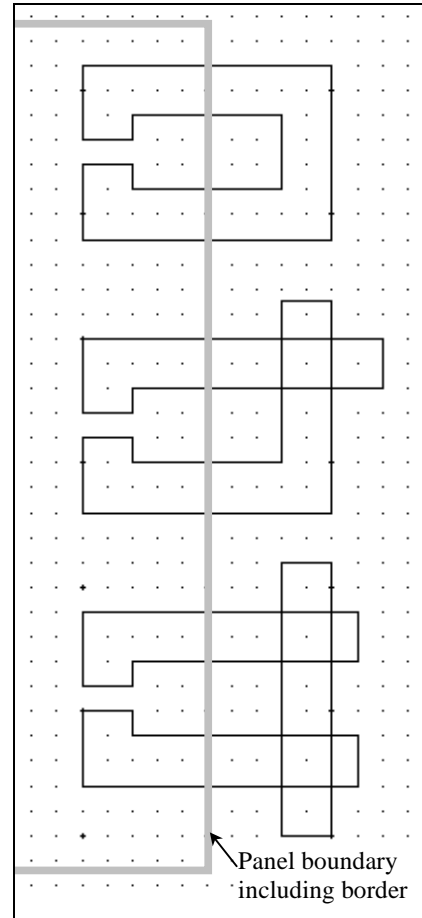


Figure 97: Bottom configuration will be processed incorrectly by the QUICK_PASS algorithm.

Due to rare cases like the one above, you must be careful not to use the QUICK_PASS option in final runs on a design.

When rules will be ignored due to the QUICK_PASS option, you must reply to a warning prompt to proceed. To avoid the warning prompt, use the ALLOW_QUICK rule or command line option.

Some rules cannot be processed at all when the QUICK_PASS option is used. The rules that indicate that extra passes are required in the table on page 125 will not be executed at all when the QUICK_PASS option is used. These rules are listed here in Figure 98.

Since CONNECT rules are not processed, this also means that the /CONN and /~CONN restrictions of the MINSPACING rule are ignored.

BRIDGE
CONNECT
ISLANDS
MAX_SPACING
OVERLAPPING
STAMP
TOUCHING

Figure 98:
Rules not executed when QUICK_PASS is used.

Effects of Panel Processing on Generated Layers

Shapes generated by the DRC (including error shapes) must be cut at the panel boundary to be stored for the next pass. Otherwise two copies of shapes that cross a panel boundary would be generated. Let us say that the rule “C = A AND B” is executed on the shapes in Figure 99. The shape on layer C will be created in both panels. To avoid creating two identical shapes on layer C, the DRC cuts the shape at the panel boundary so that each panel contains the portion of C that lies within the panel boundary.

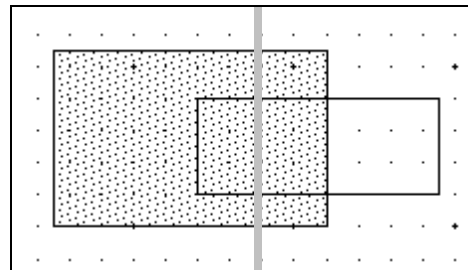


Figure 99: Shapes on layers A and B and panel boundary.

Cutting shapes at panel boundaries can lead to problems when the shapes have skewed sides. Let us say that the shapes in Figure 100 are used to generate layer C with the same Boolean rule used above. Now the panel boundary intersects a side of the layer C shape at an angle.

The DRC must cut the triangle on layer C and create two shapes as shown in Figure 101. If the dots represent the DRC resolution grid, you can see that the new vertices to create the C shapes must be shifted to lie on the grid. This results in a distortion of the shape.

This type of problem is usually resolved when the shapes are output to ICED™. The DRC grid is much finer than the ICED™ grid. Snapping all coordinates to the coarser ICED™ grid forces minor distortions like this to be removed except in rare cases. (Refer to the examples in the discussion of resolution grids on page 79.)

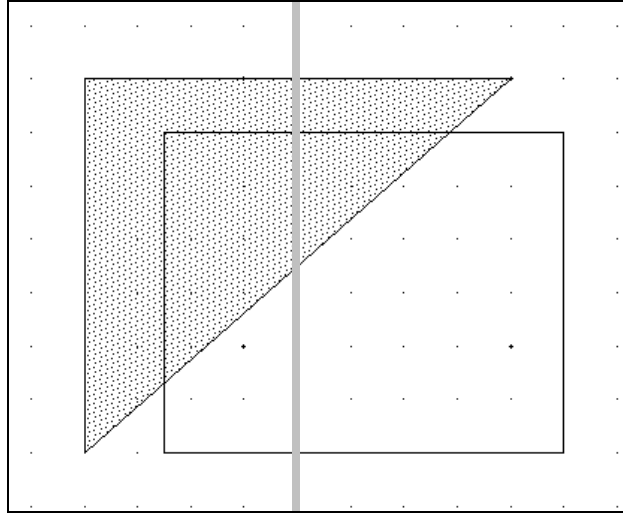


Figure 100: Shapes on layers A and B and panel boundary.

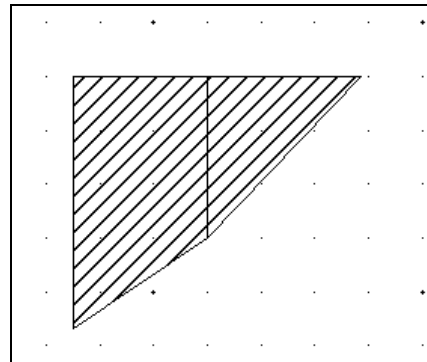


Figure 101: Layer C shapes stored for next pass.

Subject	Importance	Page
PANEL_VERTICES rule	Modify default panel size calculations based on memory available	290
PANELX and PANELY rules	Used to set panel size directly	293
NO_PANELS rule	Used to force DRC to use a single panel the size of the design	278
PANEL_X and PANEL_Y command line options	Used to override panel size set in rules file	358
BLOAT_ANGLE rule	Affects border when BLOAT or SHRINK rules are processed	311
BORDER rule	Overrides calculated border	193
BORDER command line option	Overrides calculated border on DRC command line without recompiling rules	348
SHOW_BORDER command line option	Reports border calculations in DRC log file	348
QUICK_PASS command line option	Faster algorithm that can lead to incorrect results at panel borders	337
ALLOW_QUICK rule and command line option	Avoid having to respond to a warning prompt when QUICK_PASS may miss errors. Either rule or command line option may be used. Both have the same effect.	182 338

Figure 102: References for panel processing.

Hierarchical Checking and Hierarchical Output

By default, the DRC retains the some of the hierarchical structure of cell data. While the verification rules must be processed on flat data, most of the DRC processing is performed on hierarchical data.

Unless you use a very simple rule set that executes in a single pass, the DRC must store the results of one pass for the next pass. Storing this data hierarchically reduces the storage requirements significantly.

When the DRC algorithms require flattening of cell data, only the current panel is flattened. The remainder of the data remains hierarchically nested.

To prevent hierarchical processing, you will have to add command line options to flatten cells on input. This will be covered a little later.

When you use the DRC to generate output layers, you can preserve the hierarchy. All layers will be created in cells that mimic the original cell structure. This can reduce the disk space required for the output files considerably.

Regardless of how you choose to store the output data, the DRC will by default process the data of multi-pass runs hierarchically. This conserves disk space required for the DRC scratch file and usually reduces processing time. Depending on how the design is nested, the processing time may be 10%-20% faster or slower than processing the data flat.

Hierarchical Processing Algorithm

The hierarchical processing performed on each panel during layer generation is:

Begin in the lowest level cells (those cells with no subcells). Compute the new layer in these cells first.

Go up one level in the hierarchy and temporarily flatten the nested cells. Compute the new layer and then subtract the new layer stored in the subcells.

Continue up the chain of hierarchy to the main cell. At each level in the hierarchy, the entire layer is generated from the flattened data and then the data in subcells is subtracted.

Look at Figure 103. Let us say that a shape on a wire layer is contained in the nested cell indicated with the dotted line. Two shapes on the same layer meet this cell on either side in the main cell. Now the DRC processes a shrink operation on this layer.

The shrink is performed in the nested subcell first.

Next the shrink is performed in the main cell. The entire wire is shrunk. Then the layer in the subcell is subtracted from the wire. This leaves the shapes shown in Figure 105 in the main cell.

In the entire nested design, the shrunk wire runs without a break through the cell as shown in Figure 106.

The hierarchical processing algorithm can be expensive in terms of processing time. The layer generation operations may be performed several times on each nested shape. The subtraction operations take processing time as well. However, in most designs this processing is more time

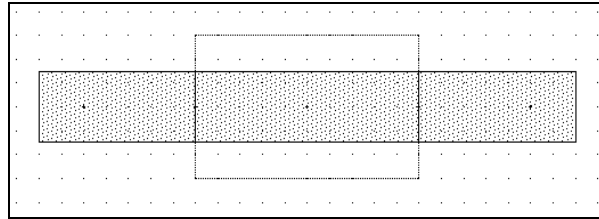


Figure 103: Wire layer in nested cell and main cell.

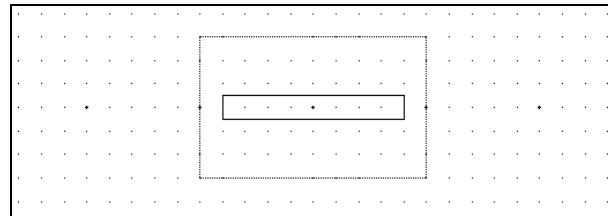


Figure 104: Shrunk layer in nested cell.

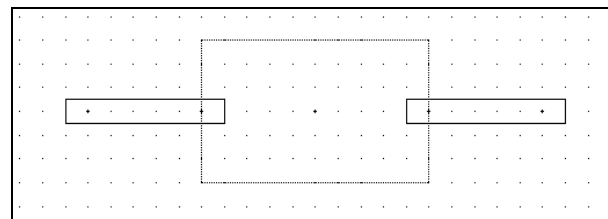


Figure 105: Shrunk layer in main cell.

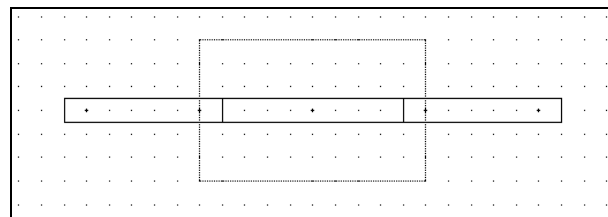


Figure 106: Shrunk layer in nested cell and main cell.

efficient than flattening the entire design before processing. The time saved in storing the design hierarchically is usually significant.

Dangerous Operations

Hierarchical processing has an important side effect. Since the DRC enforces no design constraints on how cells may overlap, the contents of a higher level cell may affect how a layer should be generated in a subcell. We refer to operations that may have this side effect as dangerous operations.

A dangerous operation is defined as **an operation that may result in the contents of a subcell being generated in error due to the contents of a higher level cell**. Dangerous operations include operations that remove material from a layer and operations that depend on touching relationships to shapes that may be in higher level cells.

For example, let us consider the simple Boolean operation " $C = A \text{ AND NOT } B$ " where a shape on layer A is in a subcell and an overlapping shape on layer B is in the main cell.

When the DRC is processing this operation, a shape on layer C is generated in the subcell that is a copy of the layer A shape. Since the DRC cannot "see" the shape on layer B, it is not considered.

When the DRC then processes the main cell, it "sees" the shape on layer B, but it is too late to change the contents of layer C in the subcell. Also, other instances of the subcell probably should not be altered.

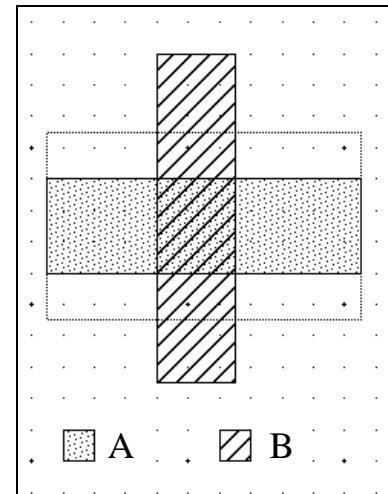


Figure 107: Layer A in subcell and layer B in main cell.

How the DRC Works: Hierarchical Checking and Hierarchical Output

The DRC will warn you at this point that the layer has been processed incorrectly. However you may prefer that the DRC generate layer C correctly in the first place. You can direct the DRC to generate the result of dangerous operations "safely".

When a dangerous operation is processed safely, the result of the operation is not generated until the DRC is sure that no incorrect results can be generated. This means that the DRC will not generate the result until it is processing the flattened main cell.

In the example above, if you direct the DRC to process the Boolean operation safely, no shapes on layer C would be generated in the subcell at all. Instead, the layer C shape would be created correctly in the main cell. No warnings would be generated. However, even shapes on layer A in subcells that are not overlapped by shapes on layer B would result in layer C shapes in the main cell.

The list of dangerous operations is provided in Figure 108. The rules that have special options listed are dangerous only when those options are used in the rule.

It is not obvious why some of these rules can result in mistakes due to dangerous processing. Let us cover a few examples.

Rule	Special options causing danger
AND	NOT keyword
ASPECT_RATIO	
BLOAT	
BOUNDS	
IS_BOX	
ISLANDS	
MIN_AREA	
NOT	
OR	NOT keyword
OVERLAPPING	NOT keyword or count restriction
RECTANGLES	
SHRINK	
STAMP	
TOUCHING	NOT keyword or count restriction
XOR	

Figure 108: List of dangerous DRC operations

The BLOAT rule can result in distortions when it is handled dangerously. **This is due to the fact that shapes in subcells that touch shapes in other cells are not merged into single shapes before an operation that is handled dangerously.**

Consider Figure 109. The crosshatched area represents two shapes on a layer that is then bloated. When these shapes are in the same cell (or when they are handled safely) they will be merged into a single shape by the DRC before the bloat. The DRC will then bloat the shape as shown in Figure 109.

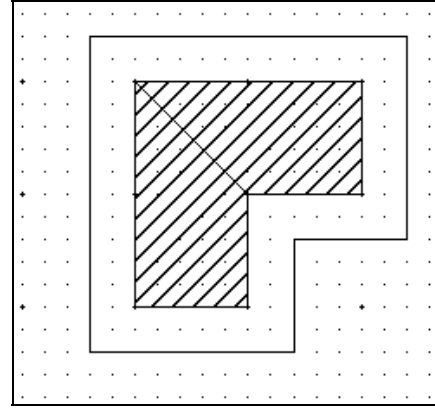


Figure 109: Result of bloat when touching shapes are merged first.

Now consider what the DRC will do if the crosshatched shape on the right is contained in a subcell and the DRC processes the operation dangerously. This shape will be bloated before the DRC is aware that a touching shape on the same layer exists. The shapes will not be merged before the bloat and the unusual bloated shapes shown in Figure 110 will be the result. The DRC would issue a warning message about the error.

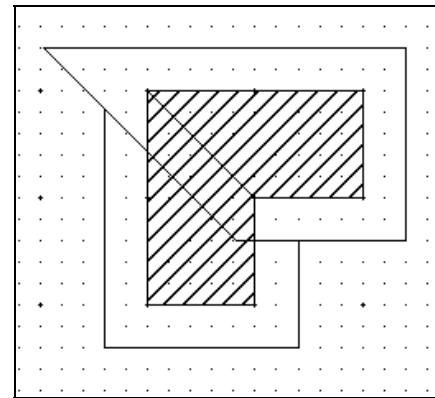


Figure 110: Result of bloat when DRC processes shapes dangerously.

The MIN_AREA rule is the only verification rule that is on the list. This is due to the fact that the DRC will process this rule before flattening the design. Touching shapes in other cells will not be merged before the area is tested. This can result in false errors, but not missed errors.

Consider the shapes in Figure 109 again. If the two crosshatched shapes are in different cells and a MIN_AREA rule is processed dangerously, the DRC may

mark either shape as an error even though the merged shape is large enough to pass the test.

The OVERLAPPING and TOUCHING rules can also have incorrect results when they are processed dangerously. However, the multi-pass algorithms used for these rules will be able to process the simplest form of the rule correctly even when they are processed dangerously. The simplest form is when a shape on one layer is tested only for touching any number of shapes on another layer. But when a count is added to the rule that restricts shapes on the result layer to those that touch a specific number of shapes on the other layer, dangerous processing can result in mistakes. If the DRC sees a shape in a cell that touches the correct number of other shapes within the cell, the DRC will copy the shape to the result layer. However, if another shape in a higher level cell touches the original shape, the original shape should not have been copied to the result layer.

The same problem arises when the NOT keyword is used in a TOUCHING or OVERLAPPING rule. In this case, a shape is copied to the result layer when it does not touch a shape on the other layer. In this case, a shape in a subcell may not touch any shapes on the other layer in the subcell, so it is copied to the result layer. But if a shape on the other layer in a higher level cell touches the original shape, it should not have been copied to the result layer.

Processing any of these rules safely will prevent mistakes at the expense of creating the result layer as one flat layer in the main cell.

There is one rule that is a special case. You would think that the CONNECT rule would have incorrect results when processed dangerously due to shapes in higher level cells. However, the DRC always processes this rule safely. Since shapes are not created by this rule, there is very little overhead to processing it safely. However, in a manner similar to the processing of the TOUCHING and OVERLAPPING rules, the DRC must process this rule in an extra pass.

One consequence of dangerous processing is that any layer generated from a layer generated dangerously is also generated dangerously. Also, any layer generated from a layer that is generated safely is also generated safely. When a layer is generated safely, the shapes are already flattened in the main cell, so there is no way to create shapes from it in the subcells.

Example: **C = A AND NOT B**
 E = C AND D

If layer C is generated safely, so is layer E. Conversely, if layer C is generated dangerously, any mistakes on that layer may be present in the E layer as well.

Oops Conditions

When a dangerous operation is handled dangerously, the DRC may generate the contents of a nested cell incorrectly. We call this an "oops condition". The DRC will realize that a shape or shapes have been generated incorrectly and issue a warning message.

For example, let us say that the Boolean operation we discussed above, "C = A AND NOT B", is processed dangerously. The DRC recognizes after the subcell has been processed that a shape on layer B in a higher level cell should subtract material from a shape on layer C in the subcell. At this point, the DRC will issue a warning message similar to the one below and continue, but further results may be incorrect.

If the DRC has recognized an oops condition, it will display a warning on your screen during the run similar to:

```
*****DANGER***DANGER***DANGER*****
*****DANGER***DANGER***DANGER*****
*****DANGER***DANGER***DANGER*****
This run may have incorrect answers---READ your
log file.
```

The log file will contain messages about specific cells similar to:

```
*****DANGER*****DANGER*****DANGER*****DANGER*****

A logical error was made processing layer C[11]
in cell MAINCELL. One of MAINCELL's subcells
contains a section of C[11] that was removed by a
logical operation in MAINCELL. This means any
```

further results in MAINCELL or a cell containing MAINCELL involving layer C[11] are likely to be wrong.

The problem can be corrected by specifying that layer C[11] or the problem subcells (not MAINCELL) be ungrouped.

An outline of the offending area (in cell MAINCELL coordinates) appears on layer 100 of error file E:\MYDIR\MAINCELL.ERR. This outline can be used to locate the subcells to ungroup.

Learn more
about subcell
error command
files on page
375.

The number in square brackets after the layer name, "[11]" in the example above, refers to the ICED™ layer number. The error file is a subcell error command file that should be executed while you are editing the indicated cell.

One method to fix the problem is to ungroup (or flatten) the subcell of the indicated cell in the layout editor. However, there are other methods you can use to fix the problem without modifying the layout. We go into these methods next.

Safe Processing Options

The **ALL_SAFE** rule will cause the DRC to process all layers safely. All shapes generated by dangerous operations will be created in the flattened main cell. Since the DRC cannot store this data hierarchically, there may be a significant increase in processing time and storage requirements. Also, if you want to generate hierarchical output, the **ALL_SAFE** rule will prevent the creation of most shapes in the subcells, so the output data will not be truly hierarchical.

You can direct the DRC to process only certain cells or layers safely. The **SAFE_CELL** rule will process only the named cells safely. Other cells will be processed dangerously. If you have received only a few danger warnings from the DRC that are caused by one or two cells, you can add this rule to your rule set to ungroup (or flatten) these cells in the DRC rather than in the layout data.

When you use the **SAFE_CELL** rule on a deeply nested cell, the dangerous layers will be generated one level up in the cell hierarchy rather than in the main cell.

The **SAFE_LAYER** rule will process only the named layers safely. This can be useful when you have only one or two layers you need to generate safely.

When the **SAFE_LAYER** rule (or the **ALL_SAFE** rule) is used on layers in a deeply nested cell, the dangerous layers will be generated in the main cell rather than one level up in the hierarchy.

Another use for the **SAFE_LAYER** rule is to process only one area of a nested design safely. You can use **INCELL** processing (see page 59) to isolate a few shapes in a cell on a new layer. Or, you can add a shape on a dummy (non-design) layer to isolate the problem area, then use one or more **AND** rules to move shapes on specific layers covered by the dummy layer to new layers. Then process only these new layers safely.

Look at Figure 111. Let us assume that this represents a large standard cell used many times in the design. The selected shape is added in the main cell to some copies of this standard cell to make it operate in a different manner.

You need to remove the intersection of these layers from layer A with an operation like "C = A AND NOT B". Even though this is a dangerous operation, it will not cause problems for most of the intersections since they are all formed from shapes in the same cell. The DRC can remove the intersections accurately and store the results in the subcell. However, the selected shape in the main cell will cause an oops condition.

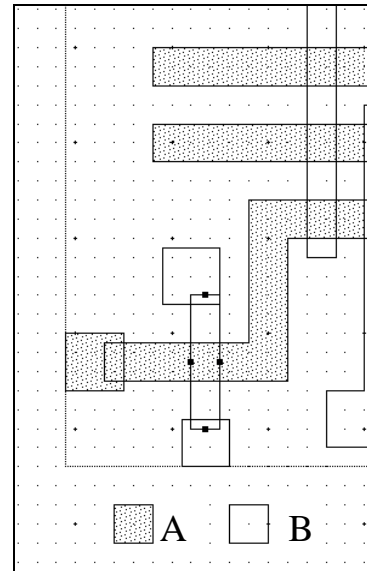


Figure 111: Standard cell with selected shape added to main cell.

If you use the **ALL_SAFE** rule or the **SAFE_CELL** rule, the entire layer will be created flat in the main cell. If you use the **SAFE_LAYER** rule on the entire layer, the result will be the same. However, if you add a box on layer **MASK** to the **subcell** such that it surrounds just this intersection you can use the following

processing to accurately generate layer C with only the area near the dangerous intersection in the main cell.

Example:

INPUT LAYER	1 A; 2 B; 110 MASK
OUTPUT LAYER	3 C
OUTPUT LAYER	3 C_MAIN
OUTPUT LAYER	0 A_MASK; 0 A_NOT_MASK

ALL_DANGER
SAFE_LAYER **C_MAIN**

A_MASK = A AND MASK
A_NOT_MASK = A AND NOT MASK

C = A_NOT_MASK AND NOT B
C_MAIN = A_MASK AND NOT B

The shape on layer MASK must be added to the subcell so that creation of A_NOT_MASK layer will not have an oops condition even though it is generated dangerously in the subcell.

Note that the C and C_MAIN layers are defined with the same layer number. The layers will be kept separate during the DRC run, but when they are output, shapes on both layers will be created on layer number 3.

Note that the ALL_DANGER rule is included in the above rule set. This rule indicates that all dangerous operations should be handled dangerously by default. The SAFE_LAYER rule overrides this default for only the C_MAIN layer.

The default for all dangerous operations must be provided by using one of the following rules or the compiler will issue an error message:

ALL_SAFE
ALL_DANGER
DANGER_CELL
SAFE_CELL

How the DRC Works: Hierarchical Checking and Hierarchical Output

The ALL_SAFE and ALL_DANGER rules set the default for all cells. The SAFE_CELL rule will process the named cells safely, but the default for all other cells is to process them dangerously. The DANGER_CELL rule will process the named cells dangerously, but all other cells will default to safe processing.

Until you are familiar with the effects of dangerous operations, use the ALL_SAFE rule in your rule set unless disk space and processing time is critical, or when safe processing disturbs the output cell hierarchy too much.

The DANGER_LAYER or SAFE_LAYER rules override the default behavior set with the other rules.

Subject	Importance	Page
ALL_DANGER rule	Default to dangerous processing for all cells and layers	180
ALL_SAFE rule	Default to safe processing for all cells and layers	181
DANGER_CELL rule	Default to safe processing for all cells except for cells listed in rule	207
DANGER_LAYER rule	Overrides safe processing for only layers listed in rule	209
SAFE_CELL rule	Default to dangerous processing for all cells except for cells listed in rule	297
SAFE_LAYER rule	Overrides dangerous processing for only layers listed in rule	299

Figure 112: References for safe/dangerous processing options.

Automatic Flattening of Cells on Input

The DRC can flatten some cells during input preprocessing before any rules are processed. This flattening is different than the temporary flattening of cells during the run for safe processing or error verification.

This flattening is performed on all layers and has the same effect as ungrouping the cells in the original layout data. Flattening on input will result in loss of cell hierarchy during the run and in hierarchical output data.

These flattening options are most useful for optimizing the data for speed and space requirements. You can tailor how small cells and cells used infrequently will be handled. Remember that data stored in its nested form will save some storage requirements, but this hierarchical processing has overhead that takes processing time. Hierarchical processing reduces run time most when the nested cells are relatively large and used many times. You may be able to realize some speed requirements if you adjust how cells are flattened.

The default DRC behavior is to flatten cells that have five or fewer components. Processing these small cells hierarchically saves very little storage space, so keeping them in their nested form is likely to make DRC runs take longer. You can override how cells with few shapes are flattened with the **CFLATTEN** option on the DRC command line.

Keeping cells that are used only once in their nested form will result in no storage savings. However they will still require processing overhead. This is why the DRC defaults to flattening cells used only once. You can modify how cells used infrequently are handled with the **NFLATTEN** command line option.

If you prefer to flatten all nested cells, use the **FLATTEN** command line option rather than using either the **CFLATTEN** or **NFLATTEN** options.

If you want cell hierarchy preserved exactly as it is in the input data, use the **NO_FLATTEN** command line option. This option is especially useful when you are generating hierarchical output. We cover more details on this subject next.

Subject	Importance	Page
CFLATTEN command line option	Controls how cells with few components are flattened on input.	353
NFLATTEN command line option	Controls how cells used few times are flattened on input.	353
NO_FLATTEN command line option	Prevents the DRC from flattening any cells on input	353
FLATTEN command line option	Causes the DRC to flatten design entirely before processing begins	352

Figure 113: References for cell flattening options.

Hierarchical Output

While the DRC stores the design data in hierarchical fashion during the run, by default it will export shapes on all output layers in flattened form. When you run the DRC command file in the ICED™ layout editor, all shapes on these layers will be created in the current cell in the coordinates of the main cell used to create the original data. However, if you use the **HIERARCHICAL** option on the DRC command line, data will be created in cells that match the hierarchical format of the input data. This can save considerable disk space when you have many output layers. When you plan to import DRC data into your design cells, this allows you to maintain the design in its hierarchical format.

The **HIERARCHICAL** command line option is used mainly when you need to import the results of complex layer processing back into your design. It has little benefit when using the DRC to only check errors. Remember that the DRC always executes verification rules on the flattened data, so rules that produce error wires (as shown in the table on page 62) will always result in shapes in the flattened main cell.

HIERARCHICAL="suffix_string"

Shapes generated for bad polygons are always located in the subcell error command files with .ERR extensions.

This is the syntax of the HIERARCHICAL command line option. The *suffix_string* is added to the end of each cell name created by the DRC command file. This allows the shapes created by the DRC to be in separate cells from your design cells.

See a complete example of this process on page 429.

These cells are created by executing the DRC command file in the ICED™ layout editor. These cells can be added to your design cells with another command file (the hierarchical cell command file) generated by the DRC. Once this command file is executed in the ICED™ layout editor, each new cell is added to the corresponding original design cell. You should inspect the cells created by the main command file before executing the hierarchical command file. Once you execute the hierarchical command file, your original design cells will be modified.

The hierarchical cell command file has a similar file name to the main command file except that the file extension is .ADD instead of .CMD. To execute the hierarchical command file, launch the ICED™ layout editor to edit a new temporary cell. You cannot execute this command file while editing one of your design cells since the command file contains EDIT commands to modify these cells. An EDIT command will fail if it cannot open a cell due to the fact that the cell is already open in the current layout editor session.

Once you have launched the layout editor to edit some new temporary cell, execute the hierarchical command file with the command:

@output_file_base_name.ADD

where *output_file_base_name* is the third parameter on the DRC command line. It is critical to include the .ADD file extension when executing this file.

The new cells added by a hierarchical command file can be time consuming to remove again once it has been executed and the cell files saved to disk. If you have added cells from a previous DRC run and need to create a new set that is slightly different, be sure to not only delete each DRC generated cell from each original design cell, but also delete the cell files generated by the previous run before executing the new command file. If cell files exist with the same names as the names in the "EDIT CELL" commands in the command file, the command file will modify these existing cell files rather than create new cells.

How the DRC Works: Hierarchical Checking and Hierarchical Output

If you specify the *suffix_string* as "", the main command file will add shapes to your original cells rather than create new cells. In this case, your original cells will be modified without warning. **This can be very risky** unless you know exactly what you are doing. Be sure to back up your design carefully before attempting to use the DRC output in this manner.

This process and the files involved are described in more detail beginning on page 372. You should read this information thoroughly before attempting to import hierarchical data.

When you do use the HIERARCHICAL command line option, you may also want to add the NO_FLATTEN command line option. This will preserve the cell hierarchy entirely instead of flattening small cells and those used infrequently.

Unless you use the ALL_DANGER rule, some shapes may be created higher up in cell hierarchy than you would expect. You will receive a warning prompt to this effect when ALL_DANGER is not used in conjunction with the HIERARCHICAL command line option. To avoid the warning prompt, use the NO_HIER_WARNING rule.

Subject	Importance	Page
HIERARCHICAL command line option	Directs the DRC to export data on output layers in hierarchical (nested cell) format	354
NO_FLATTEN command line option	Prevents the DRC from flattening some cells during input processing	353
NO_HIER_WARNING rule	Prevent warning prompt when safe processing may prevent some shapes from being created in appropriate subcells,	277
DRC command file	Description of the main command file and how to execute it in the ICED™ layout editor	365
Hierarchical command file	Description of command file to add new cells to original design cells	374
Complete example of importing hierarchical output	We strongly suggest you follow the tutorial to learn the steps for importing hierarchical data.	429

Figure 114: References for hierarchical output

Quirks of Hierarchical Processing

You may see some unusual characteristics in hierarchical data created by the DRC.

Verification rules that generate error wires are always executed on the flattened main cell. They are always processed safely. Shapes created from these rules are always created at the main cell level.

Safe processing of dangerous operations will also create shapes in cells at a higher level than you would expect. For example, let us say that you execute the rule `C = A TOUCHING B` on a design with a polygon on A in a subcell and polygon on B in a higher level cell. If these shapes touch, the shape on layer C will be generated in the higher level cell, not the subcell.

How the DRC Works: Hierarchical Checking and Hierarchical Output

A rule like the SHRINK rule may produce shapes that cross cell boundaries. The data is accurate, but it may not look the way you would expect. Wires may be shrunk away from the edges of a subcell, but connecting wires in the higher level cell will extend into the subcell area to make it up. Refer to the example on page 135.

If you do not add the NO_FLATTEN option to the DRC command line, some cells may be flattened in the output data.

The cells modified by the .ADD command file will have their environment replaced by the environment of the cell in which you execute the file. Create a temporary cell with the appropriate environment for executing the command file that modifies your original cells. See an example on page 433.

Optimizing DRC Runs

Optimizations in Rule Sets

Optimizations Performed by the Rules Compiler

The DRC rules compiler will optimize the order of execution for layer-processing rules to minimize the number of DRC passes required. Many layer-processing rules can be executed concurrently in a single pass through the design data. You do not need to worry about this issue as you write the rule set. You can locate the verification rules for a set of layers directly after the related layer-processing rules, then follow this with more layer-processing rules. Keep the rule set as readable as possible. The compiler will re-order the rule set automatically.

See an example of the compiler removing a redundant rule by compiling the Q:\ICED-\EXAMPLES-\EXAMPLE2-.RUL file.

The compiler will also remove rules that are redundant or unused. Let us say that you write several rules to generate a layer that you use in a specific rule. You then remove the rule that uses that layer. You do not need to search backward to remove the rules which created the layer. The compiler will do this automatically to optimize your rule set.

In the example in Figure 115, the rules that have been commented out with '!'s make the TEMP1, E, and F layers redundant.

```
INPUT LAYER   1 A; 2 B; 3 C; 4 D;
OUTPUT LAYER 10 E; 11 F; 12 G;
OUTPUT LAYER 101 ERR1; 102 ERR2; 103 ERR3;
OUTPUT LAYER 0 TEMP1; 0 TEMP2;

E = A AND NOT B
F = A AND NOT C

TEMP1 = BLOAT (F, 1.1)
TEMP2 = BLOAT (D, 1.1)

!ERR1 = MINSPACING (TEMP1, TEMP1, 2)
!ERR2 = MINSPACING (E, E, 3)
ERR3 = MINSPACING (TEMP2, TEMP2, .7)
```

Figure 115: Sample rule set with rules commented out.

The DRC compiler will remove the rule that creates the TEMP1 layer and will warn you about this action in the log file and console messages. The rules that create the E and F layers will remain since they are output layers. The DRC assumes that you need those layers to be exported since they are defined with valid layer numbers. However, the TEMP1 layer is just a scratch layer since it is defined with layer number 0.

Rule Subsets

The DRC itself can perform this kind of optimization at run time. If you organize your rules into rule subsets with the RULE_SET rule, this feature can make it painless to execute only a part of your rule set and have it execute as quickly as possible without a lot of unnecessary processing. When you use this feature, you do not need to edit or recompile your rules to execute only a portion of them.

When we reorganize the rule set shown above to place the commented out rules in a defined rule subset, then use the "DO=(-SET1)" option on the DRC command line, the DRC will not execute the rules that create the ERR1, ERR2 or TEMP1 rules. You can execute the DRC again at a later date without the DO command line option and execute all rules without recompiling the rules file.

The DO option can also be used to execute only specific rules by using the rule numbers indicated in the log file.

```
INPUT LAYER 1 A; 2 B; 3 C; 4 D;
OUTPUT LAYER 10 E; 11 F; 12 G;
OUTPUT LAYER 101 ERR1; 102 ERR2; 103 ERR3;
OUTPUT LAYER 0 TEMP1; 0 TEMP2;
```

```
E = A AND NOT B
F = A AND NOT C
```

```
TEMP1 = BLOAT (F, 1.1)
TEMP2 = BLOAT (D, 1.1)
```

```
RULE_SET SET1 SET2
```

```
SET1 ON
ERR1 = MINSPACING (TEMP1, TEMP1, 2)
ERR2 = MINSPACING (E, E, 3)
SET1 OFF
```

```
SET2 ON
ERR3 = MINSPACING (TEMP2, TEMP2, .7)
SET2 OFF
```

Figure 116: Sample rule set using RULE_SET organization.

Other Ways to Organize Complicated Rule Sets

Rule sets can be very long and complicated. They frequently need updating as design rules change or as they are adapted for new designs or technologies. It is a very good idea to keep them as readable and organized as possible. Use lots of comments in your rule set to allow others to follow what you have done. (You will be grateful yourself for plenty of comments if you need to update a rule set that you have not seen for a year.)

The comment indicator is the '!' character. Any text after that character, up to the end of the line, is ignored by the rules compiler.

Another way to keep rule sets concise and organized is to separate blocks of rules into separate files then use **INCLUDE** rules to combine them. This allows you to create rule set files that you may use in several different rule sets. For example, you can place rules that process the layers and test the rules for resistors in a separate file. You can then include this file with a single line in the main rules file. When a design contains no resistors, comment out the single line.

One way to make rule sets far easier to update is to use named constants instead of numbers in rules. The **CONST** rule allows you to associate a string like "M1_SPACE" with a number. You can then use the string in rules in place of typing the number. If you place all of your CONST declarations in one place (perhaps in a separate file) they are easy to find and update when technology rules change.

Subject	Importance	Page
CONST rule	Define constants you can use by name in other rules	203
INCLUDE rule	Combine separate files into one rule set	216
RULE_SET rule	Define subsets of rules that be executed without executing remainder of rule set	295
DO cmd line option	Select rule subsets to be executed	347
Rules compiler description	Complete description of rules compiler	319

Figure 117: References for optimizations in rule sets.

Testing New Rules

When writing new rules, you should test them on small test-cases before using them on a whole design. It is easy to make mistakes when writing new rules and it is much better to find these mistakes with runs that take a minute than with runs that take two hours. Carefully inspect the results of the first run on the entire design to insure that all special cases are treated in the manner they should be. It is common to encounter special cases that you did not think of when writing the rules. The NP2DS rules shown on page 71 are a good example.

You can abort the DRC once the alert is posted with the <Esc> key.

When you are testing new or modified rules, you may want to add a MAX_COUNT rule to alert you when a certain number of errors are found. The default behavior is to alert you after 1000 errors are found. This may be a high number when testing new rules. Use a smaller number to shorten a wasted run due to large numbers of false errors resulting from an error in the rule set.

If you want the DRC to stop automatically when the maximum error count is reached, add the STOP_ON_MAX_COUNT rule to your rule set.

If your new rules include dummy layer processing to isolate special cases (like resistor recognition), you should add rules to test that the dummy shapes are correctly placed. When people edit a layout, it is a common practice to blank (or hide) all but a few layers. Dummy layers that should be modified at the same time may not be visible, so they are left unmodified. If the layout has changed so that the dummy shapes are now in the wrong place, a layer may be processed incorrectly causing false errors or missed errors.

Look at Figure 118. This example was used on page 113 to demonstrate resistor recognition. RESMASK is a dummy layer used to filter the POLY layer into the RESISTOR layer and the POLY layer used as a conductive layer. If the design shapes that represent the resistor are shifted away from the dummy shape as shown in Figure 118, node 1 and node 2 will be shorted together and the resistor will be the wrong size. This could easily lead to missed errors or false errors.

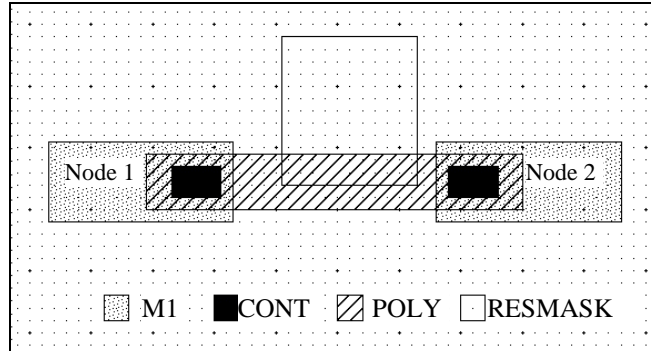


Figure 118: Shifted resistor device with dummy shape in wrong place.

You can reduce the problems caused by dummy shapes in the wrong place by adding rules that test for their proper placement. In the case above, you can add the following rules:

Example:

OUTPUT ERROR LAYER 101 BAD_RESISTOR
BAD_RESISTOR = RESISTOR NOT TOUCHING 2 POLY

If these extra rules add too much processing time to your DRC runs, you can move them to a rule set you execute less often. Just be sure that you test the dummy shapes in the final design to insure that errors are not being missed due to dummy shapes in the wrong place.

Subject	Importance	Page
MAX_COUNT rule	Change the maximum number of errors to be found before the DRC alerts the user. The default is 1000.	233
STOP_ON_MAX_COUNT rule	Stops the DRC when the MAX_COUNT number of errors has been found.	310

Figure 119: References for testing new rules.

Removing False Errors

When the DRC indicates errors that are not true design errors, we call these false errors. It is tempting to simply ignore these false errors and go on looking for the real errors. However, this is a very dangerous practice. Real errors may go unnoticed as you skip over the false errors.

You should make every effort to avoid marking these false errors in the first place. When false errors are caused by special cases of shapes on a given layer, rewrite the rules to isolate these special cases and treat them differently. Some methods of removing false errors are:

- 1) **modification of the verification rule**
- 2) **extra layer processing on the design layers**
- 3) **dummy shapes**

Various rules have different methods for avoiding false errors. The MIN_AREA and ASPECT_RATIO rules have options to modify the way shapes overlapping a panel boundary are handled. The MIN_NOTCH, MIN_SPACING, and MIN_WIDTH rules all allow you to discard errors less than a certain length. The MIN_SPACING rule has many options to discard possible errors. If you are seeing false errors for a specific rule, you should reread the syntax for the rule to see if there is a way to automatically avoid marking the false errors.

To demonstrate method number 2, let us assume that the minimum distance required to separate metal wires changes depending on the width of the wire. Wires that are 2 microns wide must be at least 2 microns apart; however wires that are only 1.5 microns wide must be only 1.5 microns apart. When you begin the design, you decide that all metal wires should be at least 2 microns apart to simplify the rules and layout. As your design progresses, you decide that you need to compress the wires in only a few places. It is tempting to leave the rules unchanged and ignore the few false errors.

If you do this, and only one real error goes unnoticed, it might be a very expensive mistake.

You can instead rewrite the rules to isolate the thin wires and test them separately. The example of separating wires by width on page 65 pairs a SHRINK rule with a BLOAT rule to isolate the thin wires. These wires are then tested with a different MIN_SPACING rule.

You can see that it may require careful thought to create rules that isolate shapes that need to be checked differently to avoid false errors. However, the extra rules are usually not that difficult to write.

If there is no way to isolate the false error shapes by layout properties, you can add dummy shapes to isolate them. (Method 3.) This should be a last resort because you will need to modify the layout to add the dummy shapes. Also, as mentioned earlier, problems often arise when the layout is modified after the dummy shapes are added.

One common source of false errors is metal letters in a corner of a chip. These errors can easily be avoided by carefully adding a rectangle on a dummy layer over the letters. Modify the rules to remove shapes covered by the rectangle from the metal layer. **Be sure to add a MIN_SPACING rule to test that the dummy layer rectangle is the minimum distance away from real shapes on the metal layer or real errors may not be found.**

Diagnosing Mysterious Errors

There are a number of things you can try when the DRC marks errors that you cannot find in the layout.

One common problem is when the DRC marks many errors that you know you have fixed since the last DRC run. If you forgot to recreate the binary layout data for the DRC with the DRC command in the layout editor after making your changes, the DRC is verifying the old data. **Be sure to always generate new binary layout data file with DRC command in the layout editor after changing the design.**

How the DRC Works: Optimizing DRC Runs

If you cannot determine which rule generated an error, you can use the `SHOW` command in the layout editor to report the tag number of the error shape. The tag number refers to the rule number that generated the shape. These rule numbers are reported in the rules compiler log file.

Detailed logging is turned on in a `MIN_SPACING` rule by adding `/DET` to the rule.

Sometimes dense areas can have many sides marked by `MIN_SPACING` rules and it is difficult to determine which sides form a pair that is too close. You can use detailed logging to list the pairs explicitly in the log file. Detailed logging is also useful when you have one unpaired error wire left over when the other error wire is discarded by a `/LENGTH` restriction.

Detailed logging can result in very large log files. It is best to use it only on small subsets of your design. When using detailed logging, run the DRC on a subcell or on small area of your design defined with one methods described next on page 159.

Occasionally, the shapes checked by the DRC are different than the shapes in the layout. If you use complicated processing (especially `BLOATS` and `SHRINKS`) to generate temporary layers for verification, tiny vertex approximations can escalate to large enough distortions of shapes to cause false errors. (See examples on page 131.) When you cannot determine why a temporary layer has errors marked, it is best to change the temporary layer to an output layer to see exactly what the DRC verified.

One related problem that is harder to diagnose is when shapes in the DRC database are marked with errors but grid resolution issues cause these shapes to be distorted or to disappear when they are output. Remember that the DRC resolution grid is much finer than the one used by the `ICED™` layout editor. Tiny slivers of shapes may disappear when the data is resolved to the coarser grid during export.

One way to resolve false errors caused by the different resolution grids is to resolve shapes in the DRC database to the grid used by the layout editor **before** they are verified. This is done with the `SNAP` and `SNAP45` rules.

If you are having trouble with a specific rule, or set of rules, and want a faster DRC run to re-execute only those rules, use the `DO` option on the DRC command line to execute only specific rule numbers. The DRC will

automatically execute all layer-processing rules that are required to execute the rules listed in the DO option.

Subject	Importance	Page
SNAP and SNAP45 rules	Resolve layers to the ICED™ layout editor grid before verification	304
Grid resolution issues	Overview of how vertices can shift on output	79
DO command line option	Execute only certain rules from a rule set	347
Limiting area checked	Diagnosing problems is much easier when checking only a small area around the problem.	159
Tag numbers	Number of DRC rule that generated a specific shape as reported by SHOW command in the layout editor	372
Detailed logging	List coordinates of pairs of sides in error in DRC log file	50

Figure 120: References for diagnosing mysterious errors

Limiting Area Checked

If you want a faster DRC run on only a specific area of your design, you can limit the design area checked. The two methods of limiting the design area are listed below.

- Use the IN keyword of the DRC command in the layout editor to restrict the input data to a small rectangle of your top-level cell. (The SEL option of the DRC command exports only selected shapes.)
- Add appropriate the LEFT, RIGHT, TOP, and BOTTOM options on the DRC.EXE command line. This method does not require you to change the DRC input file.

See page 350 to learn more about the LEFT, etc. options.

You should be aware that limiting the design area checked can lead to false errors being marked. Look at Figure 121. If the design area is limited to the area in the dashed rectangle, several false errors may be marked.

Electrical connections outside of the area boundary are ignored. Let us assume that you check the spacing of wires with a MIN_SPACING rule that prevents connected pairs from being marked as errors. The pair of wires on the bottom-right will be marked with a false error since the shape that connects them is outside of the design area checked.

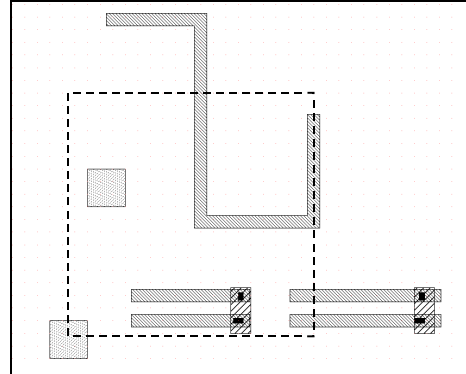


Figure 121: Design area checked is limited to dashed rectangle.

The box on the bottom-left may be marked with a false MIN_AREA violation since the shape will be cut at the boundary. This same box will also be misclassified by IS_BOX rules. The section of the long wire that is cut by the boundary may be marked with a false MIN_WIDTH violation.

Choose the boundary carefully when you limit design area checked, or ignore all errors close to the edge of the boundary. Watch for false errors caused by missing electrical connections.

Subject	Importance	Page
DRC Layout editor command	The binary data file created for the DRC can be limited to a portion of the layout.	See the Layout Editor Reference Manual.
LEFT, RIGHT, TOP, and BOTTOM command line options.	Limit area checked by the DRC	350

Figure 122: References for limiting area checked

Reducing Run Times

We just covered how to perform a much faster run when you want to zero in on a specific rule or design area, but how do you get faster run times for every DRC run?

If none of the following methods give you the speed you require, think about physical improvements. Adding more memory or a faster processor to your computer will certainly improve DRC run times. Be sure that you are not limiting the memory available to the DRC with a small number in the USE or HOG command line option. Also, be sure that the DRC is using the fastest disk drive on your computer for the scratch file. The drive used for the scratch file can be set with the SCRATCH_DIR option on the DRC command line.

Optimizing panel sizes is completely covered beginning on page 118.

The largest DRC speed improvements are achieved by optimizing the panel size and border area. Since there is a trade off between extra processing required for panel processing and time saved due the smaller amount of data stored in flattened form at any given time, time may be saved by increasing panel size or by decreasing it.

The DRC log file lists the amount of time spent by each phase of the processing near the bottom of the file. If the log file indicates that the DRC is spending significant time swapping data to disk, try reducing the panel size. If the log file indicates that little or no time is spent swapping data to disk, try increasing the panel size.

Memory Management

If you are having problems with the DRC running to completion with the memory available on your system, the first thing to try is smaller panels.

Other than panel size, the amount of memory available to the DRC is the largest factor in execution speed. You want to not only maximize the total amount of memory available to the program, but to optimize how that memory is divided.

How the DRC Works: Optimizing DRC Runs

The HOG or USE command line options limit the total amount of memory available to the DRC. (Both options perform the same function. The only difference is the units used to express the amount of memory.) If you are using the DRC in a pure DOS environment, do not use either of these options. The absence of both options allows the DRC to use all available physical memory. If you are using a multitasking operating system (such as Microsoft Windows) then use these options to reserve as much memory as possible for the DRC without impacting the other programs running on your machine.

When the DRC does not have enough memory to allocate tables or load the database, the crash messages can be somewhat mysterious. Insufficient memory is the primary suspect whenever a run crashes immediately before the log file is even created. When the system cannot provide the memory indicated by the HOG or USE parameters, the program will also crash. If a log file does get created before a crash, the amount of memory actually available to the program is listed near the top of the file.

The DRC takes the memory available to it and divides this memory into **main memory** (used for computations, tables, etc.) and **data storage** (easily swapped to/from virtual memory swap files). The default behavior is to divide the memory equally, up to a limit of 128 Megabytes of main memory.

The optimum division of memory depends on your design and the rules in your rule set. You will need to experiment with different divisions of memory to get the fastest run time.

To specify the ratio of main memory to the total amount of memory, use the `MAIN_MEMORY=main/total_ratio` command line option. To specify a fixed amount of main memory instead, use the `MAIN_USE=main_kilobytes` or `MAIN_HOG=main_megabytes` command line options.

One other option that can increase the memory available to the program for medium size runs is the `FILESIZE` option. This option limits the size of the scratch file. The DRC allocates a large virtual array page table in memory to accommodate the largest swap file possible. If you are not using a large scratch file (i.e. less than 2 Gigabytes) then you can conserve memory by reducing the largest possible size of the swap file with the `FILESIZE=scratch_megabytes` option on the DRC command line.

If you use FILESIZE to limit the maximum swap file, and your design requires more space than this, the DRC will crash with a message explaining the problem. In this case, increase the maximum size in the FILESIZE option, remove the option altogether, or optimize panel size so that less memory is required.

Rewriting Rule Sets to Improve Speed

The log file also indicates the time spent processing each rule. This data is listed by operation number. To relate the operation number to a rule, look in the rules compiler log file or add the LIST_RULES option to the DRC command line.

It can be frustrating to try to modify the rule set to make it faster. To process some rules, the DRC needs to generate special tables of data for quick algorithms. The first rule that needs this information will have the time it takes to generate the tables added to its time statistics. Other rules may use this data later without taking extra time. Therefore, if you decide to remove a rule from your rule set because it is taking a long time to process, the time it takes to create the tables may then be added to the processing time for another rule that processes the same layer.

Duplicate processing must be performed for all shapes in the border of each panel. Reducing this border by rewriting rules or separating long reach rules into a separate rule set can lead to dramatic improvements in run time.

You can see the reach required for each rule by adding the SHOW-
_BORDER
option to the DRC command line.

Remember that reach is the minimum border distance around each panel required by a rule to insure that all shapes will be processed correctly. The long reach rules force the DRC to use a large border around each panel. This large border is used by all rules in the same pass. Many short reach rules will process shapes in the border area unnecessarily.

One way to keep all rules in the same rule set, but process the long reach and short reach rules in different DRC runs, is to use rule subsets. (See page 152.) You may also be able to rewrite your rules and use clever processing to change long reach rules to short or zero reach rules.

How the DRC Works: Optimizing DRC Runs

Assume that your technology requires that all transistors must be at least 100 microns away from a pad. If you test this with a MIN_SPACING rule, the border must be at least 100 microns wide on all sides of each panel. Other rules will be forced to use this border and process all shapes in the border of each panel. A border this large may cause the DRC to fail entirely if the panels are too small.

Let us look at a clever way to process this very long reach rule. Create a rule set with a BLOAT rule to generate an output layer that contains all pad shapes bloated by 100 microns. Import these shapes into your main cell on a new layer BLOATPAD. Now when you export the entire design from the layout editor, you can test the pad rule with a Boolean rule like "ERR = BLOATPAD AND GATE". No large border is required to process this rule.

The QUICK_SPACING and QUICK_PASS Options

These DRC command line options are intended to allow you to force the DRC to use faster algorithms at the cost of the possibility of missing errors or marking false errors. These faster algorithms will be used by the DRC automatically when they cannot result in false errors as determined by the contents of the rule set and the DRC command line options. These options are available on the DRC command line to force the DRC to use these algorithms for faster intermediate runs. **Do not use these command line options on DRC runs of your final design.**

When rules will be ignored due to the QUICK_PASS option, you must reply to a warning prompt to proceed. To avoid the warning prompt, use the ALLOW_QUICK rule or command line option.

The QUICK_SPACING option can miss spacing errors in rare cases like the one shown on page 101. The QUICK_PASS option forces the DRC to execute in a single pass. This means that the DRC cannot process operations that require the DRC to recognize touching shapes like the ones shown in Figure 123. The DRC will also ignore the problems caused by panel boundaries and touching shapes. (See an example on page 130.) These restrictions may lead to some false errors and missed errors.

BRIDGE
CONNECT
ISLANDS
MAX_SPACING
OVERLAPPING
STAMP
TOUCHING
/CONN and /~CONN options of MIN_SPACING

Figure 123: Rules not executed when QUICK_PASS is used.

The QUICK_SPACING option may save on the order of 10% of DRC processing time. The QUICK_PASS option may save considerably more time. We recommend that you do not use these options on your initial runs so that you will not be misled by false errors or by the absence of errors that your rule set would have found ordinarily. However, these options can be very useful when you execute repeated runs of the DRC after minor modifications to the layout.

The Progress Report Options

During long runs, the amount of time spent by the DRC in posting messages to the console window to update the user on the progress of the run may take a significant amount of time. In one seven hour run, reducing the frequency of these console messages saved around 15 minutes of run time.

If you will not be checking the progress of the run frequently, you will lose nothing by reducing the frequency of these messages. The easiest method is to add the LONGCASE option to the DRC command line.

Newer versions of the DRC automatically suppress most progress reports for long runs based on the number of rules and the number of panels. If you want to suppress most progress reports for even shorter runs add the NO_FLASH_PANELS=*flash_limit* option to the command line, where flash limit is set to a number less than 10,000. See the option description for more details.

For medium runs, where the flash limit is not suppressing progress reports, you can reduce the refresh rate for progress updates with the DISPLAY_OPERATIONS=*min_refresh_seconds* option on the DRC command line. Setting *min_refresh_seconds* to 60 will result in the display being updated no more often than once a minute. The default is once every two seconds.

How the DRC Works: Optimizing DRC Runs

Subject	Importance	Page
Panel Processing	Complete description of how panel processing speeds DRC execution	118
Separating long/short reach rules	Example of splitting single rules file into two subsets to optimize panel border	442
Advanced tutorial	Example of optimizing panel size	445
DRC log file	Description of how DRC lists where time is spent during run	362
SCRATCH_DIR command line option	Specify disk drive for scratch file	341
USE or HOG command line options	Specify maximum total amount of memory available to DRC	339
MAIN_MEMORY, MAIN_USE, and MAIN_HOG command line options.	Specify division of memory between main memory and database memory	340
QUICK_PASS command line option	Speed DRC by eliminating multiple passes	337
QUICK_SPACING command line option	Speed DRC with quicker algorithm for MIN_SPACING rules	338
LONGCASE command option line	Optimize display for progress reports for longer runs	343
NO_FLASH_PANELS command option line	Changes definition of a “long run” so that progress reports are suppressed or not	344
DISPLAY_OPERATIONS	Changes refresh rate for progress reports	344

Figure 124: References for reducing run times

Using the DRC on Very Large Designs

Review panel processing on page 118.

When you have a large, dense design the panel size and border area become critically important. Try various panel sizes as you execute new runs. The speed improvements can be dramatic.

Review memory management on page 161.

Maximize the amount of total amount of memory available to the DRC by eliminating other programs that are appropriating memory. Set the HOG or USE command line option value to the highest possible number. When the DRC does not have enough memory to allocate tables or load a huge database, the crash messages can be somewhat mysterious. Insufficient memory is the primary suspect whenever a run crashes immediately. Once you have maximized the HOG or USE parameter, optimize the division of memory with the MAIN_MEMORY, MAIN_HOG, or MAIN_USE command line options in successive runs.

Large designs are very likely to result in large DRC scratch files. These scratch files can be as large as several Gigabytes. Be sure to add the SCRATCH_DIR option to your DRC command line to specify additional directories or disk drives. These additional directories will be used when the DRC runs out of space on the first disk drive or reaches the 2 Gigabyte file size limit imposed by the operating system. (If you used the FILESIZE option in previous versions of the DRC you may want to remove it now. The FILESIZE option is useful now only to save memory in medium size runs. The maximum size of the scratch file now grows with the number of scratch directories defined with the SCRATCH_DIR option.)

Review progress reports on page 165.

When the DRC run times are long, the LONGCASE option on the DRC command line will result in more meaningful console messages displayed during the run.

Subject	Importance	Page
LONGCASE command line option	Display more meaningful console messages during DRC run	343
USE or HOG command line options	Specify maximum amount of memory available to DRC	339
SCRATCH_DIR command line option	Specify disk drive for scratch file	341
Panel Processing	Complete description of how panel processing speeds DRC execution	118

Figure 125: References for very large designs

Preliminary Checks Vs. Final Checks

When you are in the preliminary stages of testing a large design, it is best to test the subcells first. DRC runs on small cells will execute very quickly, often in less than a minute. As errors in the smaller subcells are eliminated, proceed up the hierarchy of your design.

This method will also allow you to find problems with your rule set early in the process. It is much better to find these problems with runs that take a few minutes than with runs that take 8 hours. By the time you are verifying the entire design, your rule set should have all problems resolved.

As your rule set matures, you may separate it into different rule sets. You may need to move long reach rules to a separate rule set. If you need to have dummy layers in your design to avoid marking false errors or for device recognition, you may want to place rules that test these dummy layers in a separate set.

When rules will be ignored due to the QUICK_PASS option, you must reply to a warning prompt to proceed. To avoid the warning prompt, use the ALLOW_QUICK rule or command line option.

You may want to add the QUICK_PASS and QUICK_SPACING options to the DRC command line for intermediate runs. You will probably want to experiment with different panel sizes in intermediate runs on the whole design. If you have found an optimal panel size before you need to be making final verification runs, it will allow the final runs to be executed in a timely fashion.

Checklist for Final Run

As you proceed through the design process, it is tempting to cut corners that allow you to get faster DRC runs. You may remove rules that take too long to process when they have not found any errors in your design so far. You may add shapes on dummy layers to hide errors that you consider false errors. You will likely add the QUICK_SPACING and QUICK_PASS options to the command line. These types of methods to speed up repeated runs are not a bad idea, however you must eliminate all of these methods on final DRC runs.

In the final days of getting a major design out the door, it is far too easy to forget about the corners you cut weeks or months ago. As you make these modifications to speed things up, create a checklist for yourself that will insure

that the final design is verified without these methods that may hide errors. You may find to your sorrow that your cleverness in getting the DRC to run quickly has resulted in an undetected error added to the layout at the last minute.

The following list should only be the start of your final checklist. Keep this checklist in mind as you modify the rule set. Think hard about how changes may allow weird and unlikely layout errors to go undetected. Be sure that any of these unlikely errors will be found in the final design.

- ✓ Remove the BADPOLY=0 rule from your rule set if you have used it.
- ✓ Be sure that the NO_WARN_ACUTE or WARN_ACUTE=0 rules are not used in your final rule set.
- ✓ Remove the QUICK_SPACING and QUICK_PASS options from the DRC command line.
- ✓ If you have used the DO option on the DRC command line, remove it.
- ✓ If your layout contains shapes on dummy (non-design) layers that affect how the layout is interpreted by the DRC, remove them, regenerate them, or verify them again in the final design.
- ✓ If you have separated some rules into different rule sets (perhaps to reduce the border size), be sure to run all rule sets on the final design.

DRC Rules Syntax

General Syntax Restrictions

The syntax restrictions for DRC rule statements vary greatly from rule to rule. You must read the rule statement descriptions to determine the syntax of each rule. However, there are some general syntax restrictions which all rules have in common. We will cover these syntax issues here so we don't have to repeat them too often.

The underscore character '_', used in many keywords, is optional and can be omitted. The underscore is included for readability only and is stripped from keywords during preprocessing. However, this is done only to DRC keywords. Layer names are not preprocessed by the DRC in this manner and should be typed exactly the same way every time they are used in a rule set.

Example: **IS_BOX**
 ISBOX

Both of these ways of typing the IS_BOX keyword are equally valid.

You can use blank spaces or tabs freely between keywords and parameters in DRC rules. All extra whitespace characters are stripped by the rules compiler.

Example: **A = B AND NOT C**
 A=B AND NOT C;

These two ways of writing the AND rule are exactly equivalent. The semicolon at the end of the second rule is optional.

DRC rules are case-insensitive. This means that you can type the rule set in upper case, lower case, or any combination of the two. All text is transformed into upper case as it read by the rules compiler.

Most rules are usually typed on one line. However, when the rule is more easily read when split over several lines you can use the '&' continuation character. The '&' must be the last non-comment non-blank character on the line, and there must be at least one blank before the '&'.

Example: **ERR2 = MIN_SPACING (&
 A/OUT, &
 B/CAP=90, &
 1.1 &
 /~CROSS)**

The MIN_SPACING rule is usually written on a single line. However, when '&'s are used as in the multiline example above, the DRC reads the rule as if it were written on a single line like the line below:

ERR2 = MIN_SPACING (A/OUT, B/CAP=90, 1.1 /~CROSS)

There are several rules that allow rules to span lines without the use of '&'s. This special syntax will be indicated in the rule descriptions. Most rules that may be typed over several lines use curly brackets '{}' to enclose the text on the extra lines. When in doubt, it is valid to use the '&' in any rule.

You can add comments on lines of their own, or at the end of any line. The comment indicator is the exclamation mark '!'. Any text encountered after the exclamation mark, up to the end of the line, is ignored by the rules compiler. This means that comments can be used after the '&' continuation character.

Example: **INPUT LAYER 1 INCELL *PF C_DIFF & !Capacitor diffusion
 INCELL *NH I_DIFF & !Inductor diffusion
 NOT DIFF !All other diffusion**

Manual Notation

The syntax of individual DRC rules is described in this manual using the notation described in the ICED™ Reference Manual with one exception. Since parentheses are used so frequently in DRC rules, we will **not** use them to indicate a choice between keywords. Instead, where a choice between keywords or parameters is allowed, we will indicate this with smaller text listing the choices near the rule syntax heading.

The syntax headings use the following notation:

KEYWORD Bold type in the syntax section will be used to indicate the required rule name keyword.

parameter value Lower case italic type will be used to indicate where a value should be entered in a rule statement. The value could be a number or a string. The valid values for the parameter will be indicated in the description.

CONST *const_name* = *const_value*

The above line is used to indicate the syntax for the CONST rule on page 203. This rule is used to assign a value to a named constant that can be used in other rules instead of typing in the parameter value. When you type a CONST rule, substitute a string for *const_name* and a number for *const_value* as shown below.

CONST MY_VAL = 2.45

[**KEYWORD**] Square brackets indicate that the keyword or parameter is optional. Do not type the brackets in the rule.

result_layer = [NOT] *layer1* **AND** [NOT] *layer2*

This is the syntax description for the AND rule. The NOT keywords are optional. The parameters *result_layer*, *layer1* and *layer2* should all be replaced with layer names when the rule is typed. If the second optional NOT keyword is used, as in the following rule:

SRC_DRN = DIFF **AND** NOT POLY

the inverse of layer POLY will be used in the Boolean AND operation rather than layer POLY itself.

...

Three dots at the end of a line of sample code, or in the syntax section, indicate that the line is continued on the next line. Three dots will also precede the continuation on the next line. When you type the rule, type it all on one line without the dots or use '&'s to allow the rule to span more than one line.

Three dots in the middle of a line in a syntax description mean that several additional parameters are allowed but are not explicitly specified in the syntax description.

IS_BOX(*layer1*, *size1* [, *size2* [..., *size_n*]])

This is part of the syntax description for the IS_BOX rule. You may specify up to ten *size_n* parameters, but it would be rather verbose to list all ten parameters. The dots take the place of the missing parameters.

2_ONLY

DRC version control

2_ONLY [TOEND]

If you use the same rules set with both version 2.xx and version 3.xx of the DRC, you can use this rule to identify rules that should be executed only by version 2.xx of the DRC.

Example:

2_ONLY

NOT_RECT_A = RECTANGLES (A, (0:100, 0:100))

3_ONLY

NOT_RECT_A = NOT IS_BOX (A, (0:100, 0:100))

The RECTANGLES rule is an obsolete version of the IS_BOX rule. The IS_BOX rule is supported by version 3.xx of the DRC, but not version 2.xx. When the example above is compiled with the rules compiler for version 2.xx, only the RECTANGLES rule is executed when the compiled rule set is used by version 2.xx of the DRC. The IS_BOX rule will be ignored.

When the same rule set is compiled then executed by version 3.xx of the DRC, only the IS_BOX rule is executed.

Note that the 2_ONLY rule applies only to the next single rule in the rule set.

To identify a block of rules, add the TOEND keyword to the 2_ONLY rule. When you use the TOEND keyword, you must add an **END2** rule at the end of the block of rules.

Example:

2ONLY TOEND

NOT_RECT_A = RECTANGLES (A, (0:100, 0:100))

RECT_A = A AND NOT NOT_RECT_A

END2

3ONLY TOEND

RECT_A = ISBOX (A, (0:100, 0:100)) NOT=NOT_RECT_A

END3

When the above example is executed by version 2.xx of the DRC, the RECT_A layer will be generated as a result of the RECTANGLES rule and the AND rule. Version 3.xx of the DRC will ignore both of these rules and use the IS_BOX rule instead.

Note that the underscore is optional in the 2_ONLY, 3_ONLY and IS_BOX rules. The underscore is used for readability only. It is stripped from the keywords by the rules compiler preprocessor.

286_ONLY

DRC version control

286_ONLY [TOEND]

If you use the same rules set with different versions of the DRC, you can use this rule to identify rules that should be executed only by version 1.xx of the DRC.

The syntax is the same as the 2_ONLY rule. The TOEND keyword is used to mark blocks of rules. When this keyword is not used, the 286_ONLY rule applies only to the next single rule. See the 2_ONLY rule on page 176 for more details and examples.

3_ONLY

DRC version control

3_ONLY [TOEND]

If you use the same rules set with different versions of the DRC, you can use this rule to identify rules that should be executed only by version 3.xx of the DRC.

The syntax is the same as the 2_ONLY rule. The TOEND keyword is used to mark blocks of rules. When this keyword is not used, the 3_ONLY rule applies only to the next single rule. See the 2_ONLY rule on page 176 for more details and examples.

ALL_DANGER

Prevent cell flattening for dangerous operations

ALL_DANGER

You should read Hierarchical Checking and Hierarchical Output on page 134 to learn about dangerous operations and hierarchical processing.

When this rule is present anywhere in your rule set, the DRC will avoid flattening any cells before performing dangerous operations. This may increase or decrease DRC processing time. Dangerous operations may incorrectly process some layers. You will be warned in the DRC log file when a layer has been incorrectly processed.

You may want to use this rule when you are using the DRC to generate hierarchical output.

There are no optional parameters for this rule.

This rule is incompatible with the ALL_SAFE, SAFE_CELL, and DANGER_CELL rules. The SAFE_LAYER rule can be used in combination with the ALL_DANGER rule to override dangerous processing for certain layers.

ALL_SAFE

Force cell flattening for dangerous operations

ALL_SAFE

You should read Hierarchical Checking and Hierarchical Output on page 134 to learn about dangerous operations and hierarchical processing.

When this rule is present anywhere in your rule set, the DRC will postpone processing all dangerous operations until it is processing the flattened main cell. This may increase or decrease DRC processing time. More disk space is consumed by the scratch file when this rule is used instead of ALL_DANGER, but it will prevent the DRC from incorrectly processing some dangerous operations.

If disk space is not an issue, and you are not generating hierarchical output, we recommend that you use this rule in each rule set.

When this rule is used, all shapes generated by dangerous operations will be generated in the main cell.

The SAFE_CELL rule can be used instead to specify that only certain cells will be flattened while all others will not be flattened. See also the ALL_DANGER and DANGER_CELL rules.

The DANGER_LAYER rule can override the ALL_SAFE specification for certain layers.

ALLOW_QUICK *Avoid warning prompt for QUICK_PASS processing*

ALLOW_QUICK

Read about the QUICK_PASS option on pages 164 and 337.

The QUICK_PASS option on the DRC.EXE command line results in a much quicker execution at the cost of not processing some types of rules. This can result in missing real errors.

When you add the QUICK_PASS option to the command line and the rule set contains rules that cannot be processed by this faster method, by default you will be warned with a prompt before the DRC continues execution. You must reply to this prompt before the DRC can proceed.

However, if the ALLOW_QUICK rule is added to the rule set, the DRC assumes that you know what you are doing and will not issue the warning prompt. This can be especially useful in batch files where you want to avoid any user interaction at run time.

The DRC command line option ALLOW_QUICK performs exactly the same function as this rule in a rule set.

AND**Boolean AND of two layers**

$$result_layer = [NOT] layer1 \text{ AND } [NOT] layer2$$

This rule will create on *result_layer* the intersection of all shapes on layers *layer1* and *layer2*.

Example:

C = A AND B

To add shapes created by this rule to the error count, add the **ERROR** keyword to the **OUTPUT LAYER** rule that defines *result_layer*.

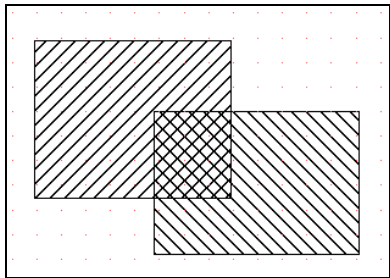


Figure 126: Polygons on layers A and B

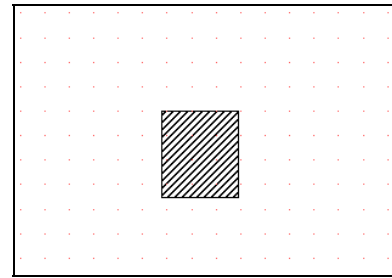


Figure 127: C = A AND B

The optional **NOT** keyword will perform the operation with the inverse of the layer.

Example:

C = A AND NOT B

See page 312 for an example where **AND** **NOT** is not sufficient to verify enclosure.

This rule will perform a Boolean AND of the inverse of layer B with layer A. In other words, layer B is used to etch layer A.

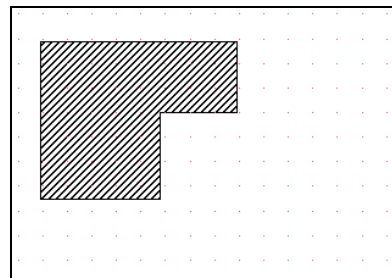


Figure 128: C = A AND NOT B

ASPECT_RATIO

Classify shapes by relative dimensions

result_layer = [NOT]⁶ **ASPECT_RATIO** (*layer1*, *max_size*,
ratio_1 [, *ratio_2* [... , *ratio_n*]]
)[NOT = *result_layer2*]⁶

A bounding box is the smallest rectangle, square with the axes, which encloses the shape. See Figure 137 on page 194 for examples.

This rule is used to classify polygons on *layer1* based on their aspect ratios. An aspect ratio is the ratio of the dimensions of the bounding box. For example, if you have a bounding box 10 units wide (in the x-direction) and 5 units high (in the y-direction), it would have an aspect ratio of:

$$\frac{10}{5} = \frac{2}{1} \quad \text{or 2 to 1.}$$

The *ratio_n* parameters are specified in the same manner as the IS_BOX and BOUNDS rules. You can enter up to ten *ratio_n* parameters. You must enter at least one. The syntax of each *ratio_n* parameter is:

(*xmin* [: *xmax*], *ymin* [: *ymax*])

To specify a simple ratio, supply a pair of real numbers separated by a comma. The first number of the pair is the relative dimension in the x-direction while the second is the relative dimension in the y-direction. The units of each dimension are the user units in the ICED™ cell.

(2,1)

This is the correct syntax to specify the 2 to 1 ratio mentioned above. This specification will find all shapes that have bounding boxes exactly twice as wide in the x-direction as they are high in the y-direction.

⁶ Only one optional NOT keyword is allowed in a single rule.

You use colons (':') to specify ranges of valid ratios. We will cover this by example further on.

See page 118
for a thorough
explanation of
panels and
borders.

The required *max_size* parameter is used specify the maximum size of a bounding box guaranteed to be classified correctly. This relates to the problem of panels and panel borders. The DRC verifies large designs one panel at a time. Shapes which cross the edge of a panel must lie within a border around the panel to be verified correctly with a rule. Usually the border is determined by calculating the reach of each rule. The reach is the minimum border required for a rule to guarantee it will process shapes which cross the border properly. Since no maximum dimension is included in this rule (only the ratio of dimensions) there is no way for the rules compiler to calculate the reach. You must specify the reach explicitly with *max_size*.

Specifying too large a value for *max_size* may slow processing. However, you should be aware that shapes with a bounding box dimension larger than *max_size* may be classified incorrectly when the DRC uses panel processing.

Example: **B = ASPECT_RATIO (A, 20, (10, 1))**

This rule will collect on layer B all shapes on layer A which have bounding boxes with an aspect ratio of exactly 10 to 1. (10 in the x-direction to 1 in the y-direction.) Shapes with a bounding box dimension larger than 20 user units may be incorrectly classified.

To expand the above rule to include shapes which have the same ratio, but which are longer in the y-direction, you need to add a *ratio_2* parameter that specifies a 1 to 10 aspect ratio.

Example: **B = ASPECT_RATIO (A, 20, (10, 1), (1, 10))**

Note that parentheses are required around each separate *ratio_n* parameter.

You can specify up to 10 *ratio_n* parameters. You may specify a range instead of an exact ratio for each *ratio_n*. You specify a range of valid ratio values in the form *min:max*.

DRC Rules Syntax: ASPECT_RATIO

Either NOT keyword is used to collect all shapes on *layer1* which do not meet the aspect ratio criteria.

Example: **B = ASPECT_RATIO (A, 20, (5:6, 1), (7, 1:2)) NOT = C**

Layer B will consist of all polygons on layer A whose bounding boxes have aspect ratios between 5 to 1 and 6 to 1 or between 7 to 1 and 7 to 2. Layer C will consist of all shapes on layer A that do not meet the criteria.

You can spread the ASPECT_RATIO rule across several lines by breaking a line after a comma. If the final NOT option is used, it must be on the same line as the closing ')'. The '&' continuation character is not required in this case.

The following example is identical to the example above.

Example: **B = ASPECT_RATIO (A, 20,
 (5:6, 1),
 (7, 1:2)
) NOT = C**

The Assignment Rule

Copy layer or inverse of layer

$result_layer = [NOT] \ layer1$

This rule is used to copy a layer or to create the inverse of a layer.

Example: **M1 = M1_IN**

This rule will copy all polygons on layer M1_IN to layer M1. This can be useful if M1_IN is an input layer that cannot be modified. The new layer, M1, can be modified as required.

The optional NOT keyword will create a layer which is the inverse of *layer1*.

Example: **NWELL = NOT PWELL**

This version of the assignment rule is exactly the same as the NOT rule described on page 281.

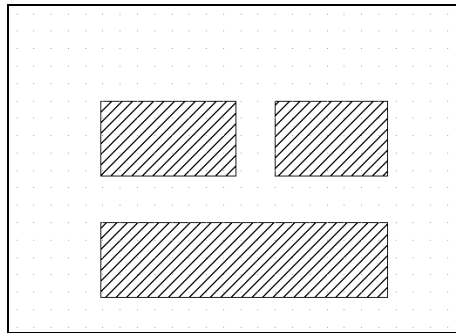


Figure 129: Layer PWELL

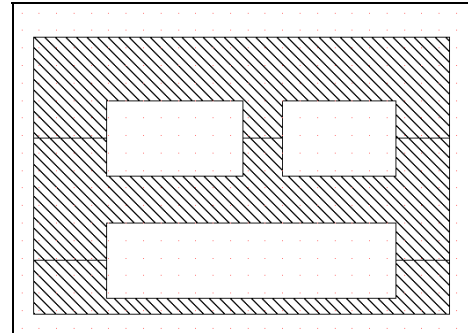


Figure 130: Layer NWELL = NOT PWELL

DRC Rules Syntax: Assignment Rule

The bounding box is the smallest rectangle, square with the axes, which encloses the design.

The rule above will create the inverse of the PWELL layer. The outer boundary of the inverse layer is slightly larger than the bounding box of your design. When the NWELL layer is used by other rules in the DRC, it will remain one large polygon with holes in it. If the NWELL layer is an output layer, before the DRC can output the layer as ICED™ components, the shape must be divided into several polygons. Polygons with holes not connected to the outer boundary are not valid components in ICED™. The somewhat arbitrary cut lines (where the NWELL shape is cut to create valid polygon shapes) will have no effect on processing in the DRC.

The CUT_RESOLUTION rule is used to define the grid for cut lines when a shape with holes is cut into valid polygon shapes.

BAD_POLY***Assign layer number for bad polygons***

BAD_POLY [=] *layer_number*

See page 74 for more details on bad polygons.

This rule allows you to change the layer number used to collect bad polygons. These are polygons that are likely to cause problems for other programs that use layout data, such as mask-processing software.

When no BAD_POLY rule is present in your rule set, the default layer number for bad polygon output is 99.

Example:

BADPOLY = 0

This rule will suppress bad polygon output. This may save space and time for preliminary checks on large input files. Warning messages for all bad polygons will still be listed in the DRC log file.

The DRC will mark bad polygons found on all input layers unless you add the NO_CHECK_INPUT rule to your rule set. When the NO_CHECK_INPUT rule is used, the DRC will ignore bad polygons found on layers that are defined but not used in the rule set.

Subcell error command file names are always created using a .ERR file extension.

When the bad polygon layer number is non-zero, bad polygons will result in shapes on that layer number in the subcell error command files. See page 375 for details on subcell error command files.

BLOAT

Expand shapes

$$result_layer = \mathbf{BLOAT} \ (layer1, offset_val)$$

To shrink polygons see the SHRINK rule on page 302.

Use the BLOAT rule to expand polygons on *layer1* and store them on *result_layer*. All sides of the polygons will be shifted outwards in a parallel manner by *offset_val*. *offset_val* must be a positive real number in ICED™ user units.

Example:

A = BLOAT (B, 1.2)

See an example of using a pair of SHRINK and BLOAT rules to classify shapes by size on page 64.

Note that the parentheses and comma are required in the BLOAT rule.

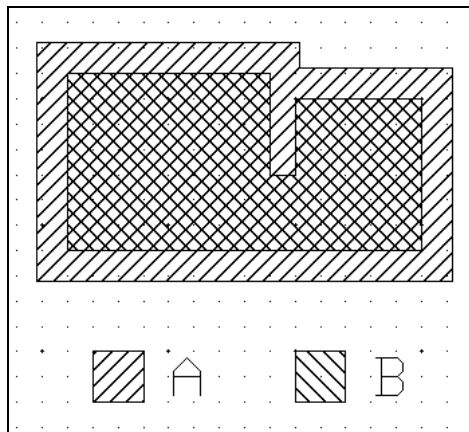


Figure 132: Note that the notch in layer B disappears after bloating.

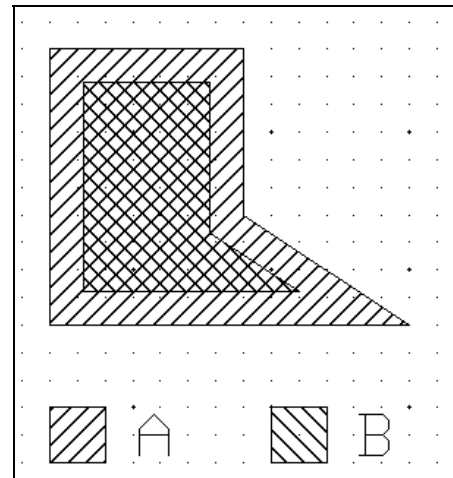


Figure 131: A = BLOAT (B, 1.2)

Bloating can remove features of complex polygons. Notches or holes can disappear.

If you are using BLOAT on polygons with acute angles, you should refer to the BLOAT_ANGLE rule on the next page for important information on the side effects of bloating sharp angles.

BLOAT_ANGLE

Define angle for BLOAT rule

BLOAT_ANGLE = *bloat_angle*

If you do not have acute angles in your design, you should not use the BLOAT_ANGLE rule. The default of 44.9° will prevent excessive run times.

This rule is important only if your layout contains polygons with sharp points or notches. It controls how shapes with sharp angles are bloated or shrunk. Points with angles more acute than *bloat_angle* will be blunted before they are bloated. The *bloat_angle* parameter must be a real number in the range 1:179. When the BLOAT_ANGLE rule is not used, the DRC uses a default of 44.9° for the bloat angle.

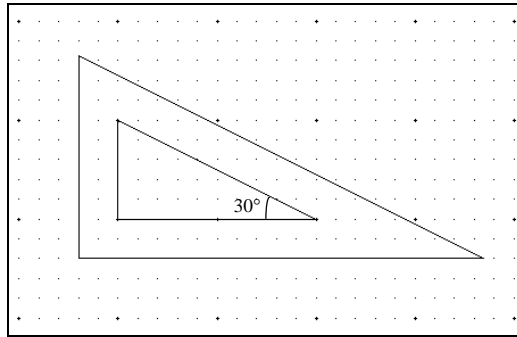


Figure 133: Unconstrained bloat of a 30° angle.

To see why bloating sharp points can be a problem, see Figure 133. The inner triangle has a sharp point with an acute angle of 30°. If this bloat is not constrained, the bottom dimension of the polygon will more than double when it is bloated by an *offset_val* of 2.

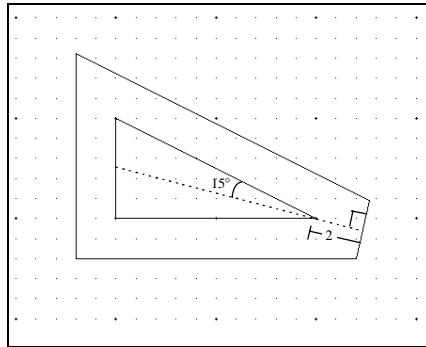


Figure 134: Constrained bloat of a 30° angle.

To avoid this type of expansion for relatively small bloats, the DRC defaults to constraining bloats on any angle less than 45°. The bloated shape is cut by a line perpendicular to the line that bisects the acute angle. The cut will be made at a distance equal to the bloat *offset_val* along the bisecting line. It is as though the point at the acute angle is blunted by an infinitesimal line segment before the bloat.

DRC Rules Syntax: BLOAT_ANGLE

If this is not how you want your acute angles bloated, you must use the BLOAT_ANGLE rule in your rule set. Set *bloat_angle* to a small enough angle to remove the constraint for critical polygons. However, you should be aware that as the bloat angle gets smaller, the DRC run time gets longer. This is due to panel processing and borders which is a subject not covered here. To understand how the bloat angle affects run times, see page 126.

The bloat angle affects the SHRINK rule as well, since a shrink is really processed as a bloat of the inverse of a layer.

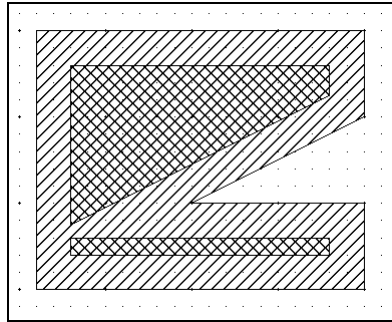


Figure 135: Unconstrained SHRINK of polygon with acute angle notch.

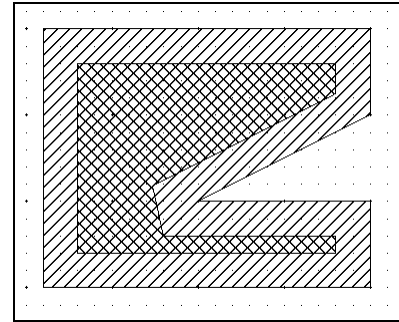


Figure 136: Constrained SHRINK of same polygon.

You can use the BLOAT_ANGLE rule more than once in a rule set.

Example:

```
B = BLOAT ( A, 2 )
BLOAT_ANGLE = 10
C = BLOAT ( A, 2 )
BLOAT_ANGLE = 44.9
D = BLOAT ( A, 5.1 )
```

You can see the bloat angle used for each BLOAT rule in the rules compiler log.

In the example above, the default bloat angle of 44.9° will constrain the bloats that create layers B and D. A bloat angle of 10° will constrain the bloats that create layer C. However, since one of the layers uses such a small bloat angle, the run time will be much longer than if all layers used the default.

BORDER

Explicitly define panel overlap

BORDER [=] *border_dimension*

It is very important to read Panel Processing on page 118 before using this rule.

Use this rule to override the panel border calculations that the DRC performs and set the panel border directly. This rule should be used only when you need to set the panel border to a smaller dimension than that calculated by the DRC. **Make sure that you know what you are doing before you use this rule.**

It is dangerous to set the panel border to a value smaller than the value calculated by the DRC. **You can cause real errors to be missed since the reach of some layers will now be greater than the border.**

The BORDER option on the DRC command line (see page 348) overrides this rule.

BOUNDS

Classify shapes by the size of their bounding box

$result_layer = [NOT]^7 \text{ BOUNDS } (layer1, size1 [, size2 [..., sizen]]$
 $) [NOT=result_layer2]^7$

The ASPECT-RATIO rule classifies shapes by the ratio of bounding box dimensions.

This rule is very similar to IS_BOX, except that the size criteria applies to the bounding box of any shape rather than only to the dimensions of rectangles. The bounding box of a shape is the smallest rectangle, square with the axes, which will enclose the shape. The bounding box of a rectangle square with the axes has the same dimensions as the rectangle itself.

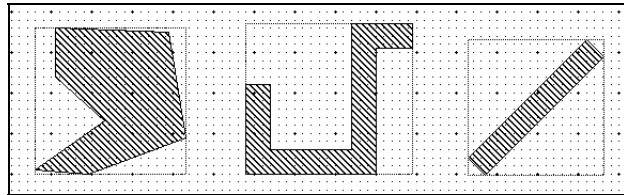


Figure 137: Bounding boxes of non-rectangular shapes.

The syntax of the *sizen* parameters, and the use of the optional NOT keywords, is exactly the same as the IS_BOX rule. See that rule (starting on page 222) for more examples. You can enter up to ten *sizen* parameters. You must enter at least one. The syntax of each *sizen* parameter is:

$(xmin [: xmax], ymin [: ymax])$

When you supply a simple pair of real numbers separated by a comma, the first number of each pair is the dimension in the x-direction while the second is the dimension in the y-direction. When the maximum values are not included, they are assumed to be equal to the minimum. The units of each dimension are the user units in the ICED™ cell.

⁷ Only one optional NOT keyword is allowed in a single rule.

Example: **B = BOUNDS (A, (10,5))**

This rule will collect on layer B all shapes on layer A which have bounding boxes 10 units wide in the x-direction and 5 units high in the y-direction.

The BOUNDS rule can be used to filter out large or small shapes on a layer rather than collect shapes of an exact size. Use the ratio form of the *sizen* parameters. You must enter two *sizen* parameters with an upper bound on the x and y larger than the largest dimension of a shape on *layer1*.

Example: **B = NOT BOUNDS (A, (0:4,0:10000), (0:10000,0:4))**

This example will create on layer B all shapes on layer A which have **both** bounding box dimensions greater than 4 units. Any shapes with either bounding box dimension less than or equal to 4 units will not be copied to layer B. This example assumes that no shape on layer A will have a dimension larger than 10,000 units.

Example: **B = NOT BOUNDS (A, (0:4,0:4))**

This example will create on layer B all shapes on layer A which have **either** bounding box dimension greater than 4 units.

You can spread the BOUNDS rule across several lines by breaking a line after a comma. If the final NOT option is used, it must be on the same line as the closing ')'. The '&' continuation character is not required in this case.

Example: **SMALL_A = BOUNDS (A,
 (0:4,0:10000),
 (0:10000,0:4)
) NOT = LARGE_A**

The example above will copy to the SMALL_A layer all shapes on layer A that have either bounding box dimension less than or equal to 4 units. All shapes that have both bounding box dimensions greater than 4 units will be copied to the LARGE_A layer instead.

BRIDGE*Recognize air bridges*

```
BRIDGE {  
    BRIDGE = bridge_layer  
    POSTS = post_layer  
    LENGTH = min_length [: max_length ]  
    WIDTH = min_width [: max_width ]  
    IS_BRIDGE = result_layer  
    NOT_BRIDGE = result_layer_2 } must use one,  
                                     can use both  
    [ L/W = min_ratio [: max_ratio] ]  
    [ POINT_TOLERANCE = tolerance_1 ]  
    [ POST_TOLERANCE = tolerance_2 ]  
}
```

Shapes created on the result layers of this rule are not automatically counted as errors unless you add the ERROR keyword to the OUTPUT LAYER rules that define the layers.

The BRIDGE rule is used to find air bridges. If you don't know what an air bridge is, it is unlikely that you will ever need this rule. It is of interest primarily to users of the Gallium Arsenide technology.

The BRIDGE, POSTS, LENGTH, and WIDTH keywords are required. At least one of the IS_BRIDGE or NOT_BRIDGE keywords must be used. You can use both. The other parameters are optional conditions that must be met for the shapes on *bridge_layer* to be considered air bridges.

To be a valid air bridge, a polygon must meet the following three conditions:

- 1) The polygon on *bridge_layer* must be rectangular. The rectangle does not need be square with the axes.
- 2) The polygon on *bridge_layer* must share opposite end-sides with polygons on *post_layer*, one at each end. Each end-side of the air bridge must be coincident with a side of the post. The post may be wider than the bridge, but the entire end-side of the bridge must touch the post.

- 3) The bridge must fall within a certain range of lengths, widths, and aspect ratios (length/width). The length is the distance between end-sides shared with a post. The width is the distance between the other two sides.

When the `IS_BRIDGE` keyword is used, the *result_layer* will contain all shapes on *bridge_layer* that meet the air bridge criteria. When `NOT_BRIDGE` is used, *result_layer_2* will contain all shapes on *bridge_layer* that are not air bridges.

When entering the `LENGTH` and `WIDTH` parameters, you can enter either a single size or a range. To enter a range, use a colon (':') to separate the maximum value from the minimum value.

Example:

```

INPUT LAYER      5 METAL; 6 POST
OUTPUT LAYER     38 BRIDGE_OUT
SCRATCH LAYER    BRIDGE_IN

```

```

BRIDGE_IN = METAL AND NOT POST
BRIDGE {
    BRIDGE = BRIDGE_IN
    POSTS = POST
    WIDTH = 2
    LENGTH = 5:20
    IS_BRIDGE = BRIDGE_OUT
}

```

This set of rules will recognize bridges from 5 to 20 units long where layer METAL is crossed by shapes on layer POST. Since the BRIDGE rule requires bridge shapes to share sides with the post shapes, we have added the AND rule to etch the METAL layer with the POST layer. The layer BRIDGE_OUT will contain rectangles for bridges 1 and 2 as shown in Figure 138.

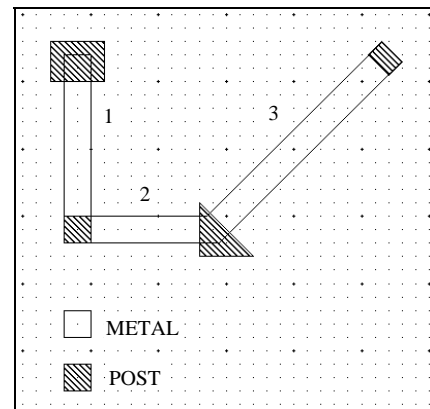


Figure 138: 3 Air bridges.

Candidate 3 in Figure 138 is a special case since it is not square with the axes. You cannot predict the exact length or width of air bridges that are not square with the axes due to vertex approximations. Unless all air bridges are horizontal or vertical, enter a range of lengths and widths. To modify the above rule to recognize candidate 3 as a valid air bridge, change the WIDTH parameter to:

WIDTH = 1.99 : 2.01

Use the $L/W = \text{min_ratio} [; \text{max_ratio}]$ option to add an additional length to width ratio constraint. You can enter a single ratio by using only *min_ratio*, or specify a range by using *max_ratio* as well. Either ratio can be entered in fraction form (e.g. "5/3") or as a single number in decimal form (e.g. "1.6667").

Example: **L/W = 5/1 : 6/1**

Add this parameter to the BRIDGE rule to restrict valid bridges to those with aspect ratios between 5 to 1 and 6 to 1. Adding this parameter to the BRIDGE example above will result in only bridge 1 (see Figure 138) on the BRIDGE_OUT layer.

The POINT_TOLERANCE = *tolerance_1* option defines the spacing tolerance for the corners of the air bridge. This accounts for small round-off errors in air bridge corners where the air bridge is not square with the axes. Each corner of the bridge must be within *tolerance_1* units in both the X and Y directions of where it would be if the bridge were exactly a rectangle.

Use the POST_TOLERANCE = *tolerance_2* option to allow a small overlap or misalignment between the post sides and the bridge layer sides. The point at one end of a *bridge_layer* side must be within *tolerance_2* units in both the X and Y directions of the equivalent point on the post edge. However, if the bridge shape does not touch the post shape, the bridge shape will not be considered a bridge. The touching criterion (i.e. the bridge shape must touch 2 shapes on the post layer) must be met before the BRIDGE rule will examine the other criteria to determine if the shape is a bridge.

You can get a report in the DRC log file on the **default** tolerance used for both specifications by adding SHOW_SCALES to the DRC command line. It is listed as "Smooth-_tolerance"

Both of these tolerances default to small non-zero numbers (usually .001 user units, but they may increase slightly for very large designs). The default tolerances are usually sufficient to recognize air bridges when the geometry varies slightly due to resolution grid rounding of the coordinates.

You can enter more than one parameter on a line if you separate the parameters with commas.

CONNECT

*Electrically connect layers***CONNECT** *layer1 layer2* [**BY** *layer3*]

See the STAMP rule to form electrical connections to layers that are poor conductors.

The CONNECT rule will form electrical connections between touching shapes on the given layers. All shapes that are electrically connected will be considered the same node and will be assigned the same node number.

In the DRC, the electrical connections defined by this rule are only used by the MIN_SPACING rule when the /CONN or /~CONN options are used.

See Electrical Connections on page 110 for more important information and examples of electrical connections.

The touching criterion for the CONNECT rule is the same as that used for the TOUCHING rule. Two shapes are considered touching if they share a finite area or if their edges share a finite length. Shapes that touch only at a point are not considered electrically connected.

When the BY keyword is not used, shapes on *layer1* which touch shapes on *layer2* are considered to be electrically connected.

Example:

CONNECT M1 M2

CONNECT rules are not processed when the QUICK_PASS option is included on the DRC command line.

When this CONNECT rule is used, any shape on M1 which overlaps or shares a finite portion of an edge with a shape on M2 will be considered electrically connected to the shape on M2. If this rule was executed on the shapes in Figure 139, the shape on M2 and the top two wires on M1 would all be electrically connected and stamped with the same node number. The bottom wire on M1, which touches M2 only at a point, would be a separate node.

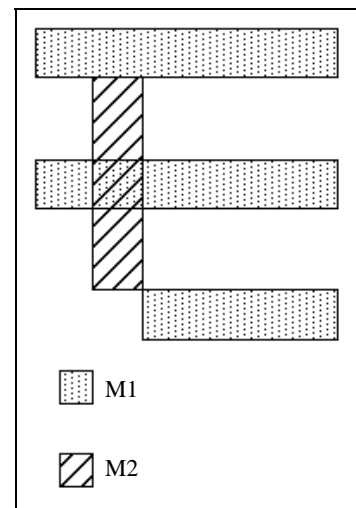


Figure 139: The top two M1 wires will be electrically connected to the M2 wire.

The BY keyword is used to simulate connections between layers that are formed by vias or other contact layers. When the BY keyword is used, the touching criterion changes. For a shape on *layer1* to be electrically connected to a shape on *layer2*, the shape on *layer1*, the shape on *layer2*, and a shape on *layer3* must all share a common area.

Mere touching or overlapping of these layers is not enough to connect the shapes.

Example:

CONNECT M1 M2 BY VIA

See an example that includes CONNECT rules on page 402.

When the above rule is used to connect the M1 and M2 layers shown in Figure 140, only the M1 wire with the label "THREE" will be connected to the vertical M2 wire. Wire "ONE" overlaps the M2 wire, and both touch the via shape, but the via shape does not overlap the common area where the metal layers overlap. Wire "TWO" fails to connect for the same reason even though the via overlaps both wires. Wire "FOUR" does not overlap the M2 wire at all, so it does not connect to it even though the via shape overlaps both.

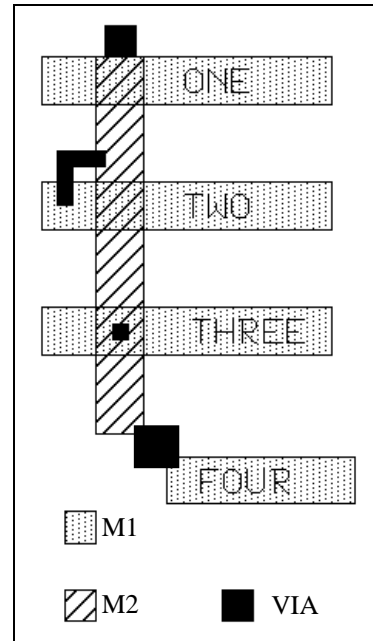


Figure 140: Only M1 wire THREE is connected to the vertical M2 wire.

You combine layer generation rules with CONNECT rules to simulate the fabrication process and electrical connectivity. You may need to process conductive layers carefully before adding the CONNECT rules.

You cannot modify a layer after it is used in a CONNECT rule. This restriction is enforced by the rules compiler. If a layer could be modified after being used in a CONNECT rule, there would be no way to guarantee that the electrical connections made by the CONNECT rule would be valid by the time the MIN_SPACING checks are run.

DRC Rules Syntax: CONNECT

The fact that polygons cross panel boundaries requires the DRC to add CONNECT rules to your rule set to connect shapes that cross panel boundaries. These rules are added automatically by the rules compiler. In the rule compiler log, they are indicated by the keyword "Generated" instead of the source line number. (See page 128 for more information.)

CONST

Define constant value

CONST *const_name* = *const_value*

or

```
CONST {  
    const1_name = const1_value  
    ⋮  
    constn_name = constn_value  
}
```

You can use a CONST rule to define a certain number as a constant that you can refer to by name in other rules rather than typing the number itself. The *const_value* must be a real number. You may **not** use exponential notation (e.g. 1.478E-9) when typing *const_value*. You may not use layer names or other strings for *const_value*.

Example: **CONST M1_EXPANSION_VALUE = .246**
 M1 = BLOAT (M1_IN, M1_EXPANSION_VALUE)

The CONST rule above defines the string "M1_EXPANSION_VALUE" as a constant with the value .246. When this constant is used in the BLOAT rule, the rules compiler will substitute ".246" for the string "M1_EXPANSION_VALUE".

You can have many CONST rules in your rule set. This allows you to define technology dependent parameters together in one place where they are easy to find and edit. When the CONST rule is not used, it will be difficult to update an old rule set with new values since they will be scattered through the rule set.

DRC Rules Syntax: CONST

If you have multiple constants to define, you can use the multiple line syntax to define all of them with a single CONST rule. Place each constant definition on it's own line, or separate definitions on the same line with semicolons. Surround the lines with curly brackets.

Example: **CONST {**
 M1_EXPAND = .246
 M2_EXPAND = .246
 MIN_M1_W = 1.9
 MIN_M2_W = 2.1
 }

Example: **CONST {**
 M1_EXPAND = .246; M2_EXPAND = .246
 MIN_M1_W = 1.9; MIN_M2_W = 2.1
 }

Both of these constant definitions are equivalent.

CUT_RESOLUTION

Place cut lines on specific grid

CUT_RESOLUTION = *grid_resolution*

See page 79 for more information on how output shapes are affected by the different resolution grids in ICED™ and the DRC.

At the end of the DRC run, when the DRC creates the shapes on output layers for reading into an ICED™ cell, some shapes may need to be cut at arbitrary locations. (See examples on page 77.) This occurs in the following cases:

a shape is cut by a panel boundary,

a shape contains a hole,

a shape has more than 199 vertices,

or

a shape has been generated from the inverse of a layer.

This rule allows you to force the cut lines to lie on a grid of your own choosing.

If you are not generating output layers for use as mask layers, but using the DRC to only check for errors, this rule is not important. If you are generating layers that will be used as design data, add this rule to your rule set with a *grid_resolution* at least as large as the resolution grid of your ICED™ cells. Express *grid_resolution* as a real number of user units.

Example:

CUT_RESOLUTION = .1

When this rule is present in your rule set, shapes that are cut for the reasons above will be cut on a .1 user unit grid.

The default resolution for the cuts mentioned above, when the CUT_RESOLUTION rule is not used, is zero. This means that the shapes that are cut may have vertices that will be rounded by some post processing software.

DRC Rules Syntax: CUT_RESOLUTION

This rule does not affect the vertices of shapes during the execution of the DRC rule set, or the vertices of shapes created by bloats, shrinks, or intersections of slanting lines. Use the SNAP or SNAP45 rules to control the resolution grid for the results of these operations.

DANGER_CELL***Prevent cell flattening for dangerous operations***

DANGER_CELL *cell_name* [*cell_name_2* [...*cell_name_n*]]

You should read the information beginning on page 134 to learn about dangerous operations.

This rule specifies certain cells that the DRC should **not** flatten when performing dangerous operations. When this rule is used, all cells not listed as danger cells will be flattened by the DRC for dangerous operations.

This rule is used primarily when you are generating design layers for hierarchical output data.

Example:

DANGER_CELL SUBCELL

This rule will prevent the DRC from flattening the cell SUBCELL for dangerous operations. All other cells will be flattened for dangerous operations.

You can supply more than one DANGER_CELL rule. You can also specify more than one cell in the DANGER_CELL rule. Simply list all required cell names without commas on the same line. If you prefer, you can use curly braces to allow more than one line of cell specifications in a single rule.

Example:

**DANGER_CELL XCELL YCELL
DANGER_CELL ZCELL****DANGER_CELL XCELL YCELL ZCELL****DANGER_CELL {
 XCELL YCELL
 ZCELL
}**

All three of these danger cell specifications are equivalent.

DRC Rules Syntax: DANGER_CELL

See an example
of wildcard
syntax on page
297.

The *cell_name* parameters can contain wildcard characters ('*'). When an asterisk is present, the DRC will handle as a danger cell any cell with a name that matches the given string with one or more characters replacing the asterisk.

This rule is incompatible with the rules ALL_DANGER, ALL_SAFE, and SAFE_CELL. When DANGER_CELL is used in combination with SAFE_LAYER or DANGER_LAYER rules, the SAFE_LAYER or DANGER_LAYER rules take precedence.

DANGER_LAYER***Override cell flattening for certain layers*****DANGER_LAYER** *layer1* [*layer2* [...*layern*]]

You should read Hierarchical Checking and Hierarchical Output on page 134 to learn about dangerous operations and hierarchical processing.

Use this rule to specify layers that should be created hierarchically (i.e. nested in subcells) rather than as flattened layers in the main level cell. This rule overrides the default specification for all cells defined by the ALL_SAFE, SAFE_CELL, or DANGER_CELL rules.

Specify the names of layers that should be generated dangerously. You cannot specify input layers in this rule. Only the layer(s) specified in this rule will be processed dangerously. Other layers in cells that contain the indicated layers will not be affected.

You may want to use this rule rather than ALL_DANGER or DANGER_CELL when you have only a small area of a large cell you need to be handled dangerously. You can add a small shape on a dummy layer that isolates the problem shapes on a new layer that you specify in a DANGER_LAYER rule. See an example on page 142.

You can supply more than one DANGER_LAYER rule. You can also specify more than one layer in the DANGER_LAYER rule. Simply list all required layer names on the same line. If you prefer, you can use curly braces to allow more than one line of layer specifications in a single rule.

Example:

```
DANGER_LAYER A B
DANGER_LAYER C

DANGER_LAYER A B C

DANGER_LAYER {
    A B
    C
}
```

All three of these danger layer specifications are equivalent.

DETAIL

Turn detailed logging on or off

DETAIL ON and DETAIL OFF

The /DET or /~DET options in the MIN_NOTCH, MIN_WIDTH, and MIN_SPACING rules override the logging mode set with this rule.

Use these rules to specify whether or not the DRC should add detailed error messages to the log file for each error found by the MIN_NOTCH, MIN_WIDTH, and MIN_SPACING rules. (These are the only rules that produce detailed error messages in the log file.) These error messages can use up considerable disk space for large designs.

The detailed logging mode is off by default until you turn it on in a specific rule or by using DETAIL ON.

The detailed logging of error messages can be turned on and off several times during a rule set. We suggest that you turn logging on only for small designs, or only for small subsets of rules when you cannot determine the exact errors from the error wires in ICED™.

Example:

DETAIL ON

RESULT1 = MINSPACING (A, A, 20)

DETAIL OFF

RESULT2 = MINSPACING (A, B, 5)

RESULT3 = MINSPACING (A, C, 2 /DET)

See page 50 for an example of detailed error messages.

When this set of rules executes, error messages with coordinate data will be printed in the log file for each violation of the rules that create layers RESULT1 and RESULT3.

(If you have a /LENGTH=*length* option in a MIN_SPACING rule, and detailed logging is enabled, the log file **will** contain details on error wires that have been discarded due to the length restriction.)

HOLE_AREA_FRACTION

Classify polygons with holes

```

result_layer = [NOT]8 HOLE_AREA_FRACTION ( ...
    ... layer1, min_fraction, max_fraction ...
    ... [/BORDER=max_size] ...
    ... ) [NOT = result_layer_2]8

```

To simply find all holes in a layer, see the ISLANDS rule.

Use this rule to classify polygons on *layer1* by the fraction of their outline area that is removed by enclosed holes. (The outline area is the total area of the polygon including the area covered by holes.) Specify the minimum and maximum fraction of the outline area that can be covered by holes as positive real numbers between 0.0 and 1.0.

Remember that the DRC merges touching polygons as a preprocessing step, so ICED™ polygons that merge to form a shape with an enclosed hole are treated in exactly the same way as polygons drawn with an enclosed hole.

Example:

RESULT = HOLE_AREA_FRACTION (A, 0.25, 1)

This rule will copy to the RESULT layer all polygons on layer A that contain holes that remove at least ¼ or 25% of the total outline area.

If the rule above was executed on the shapes in Figure 141, the two shapes on the left will be copied to RESULT (including the shape formed by two polygons that merge to form a single polygon with a hole that is exactly ¼ of the merged outline area). The shape without a hole is not copied, and neither is the shape with the hole that covers less than ¼ of the total outline area.

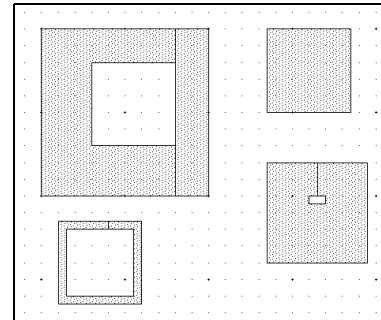


Figure 141: Two shapes on the left are copied to RESULT.

⁸ Only one optional NOT keyword is allowed in a single rule.

DRC Rules Syntax: HOLE_AREA_FRACTION

Example: **A_MED_HOLES = HOLE_AREA_FRACTION (A, 0.25, 0.49999)**
 A_BIG_HOLES = HOLE_AREA_FRACTION (A, 0.5, 1)

These two rules will classify polygons on layer A and create 2 new layers. A_MED_HOLES will contain copies of shapes on the A layer that have holes that cover at least $\frac{1}{4}$, but less than $\frac{1}{2}$, of the total outline area. A_BIG_HOLES will contain copies of shapes on the A layer that have holes that cover at least $\frac{1}{2}$ of the total outline area.

Using a minimum fraction of 0 will add shapes without holes to the shapes on the *result_layer*. To find all shapes with holes on a given layer, use a rule similar to the following with a very small but non-zero *min_fraction*.

Example: **A_H = HOLE_AREA_FRACTION (A, 0.00001, 1)**

Using the NOT Keywords

Use either NOT keyword to copy all shapes on *layer1* that do **not** meet the hole fraction criteria to a result layer. Only one NOT keyword is allowed in a single rule.

Example: **A_H = HOLE_AREA_FRACTION (A, 0.00001, 1) NOT = A_NO_H**

Adding "NOT = A_NO_H" to the previous example results in all shapes with no holes being copied to the A_NO_H layer.

If you need to collect only shapes that do not meet the hole criteria, use the first NOT before the HOLE_AREA_FRACTION keyword as in the example below.

Example: **A_NO_BIG_HOLES = NOT HOLE_AREA_FRACTION (A, 0.5, 1)**

This example copies to the A_NO_BIG_HOLES layer all shapes on layer A that do not have holes that cover at least $\frac{1}{2}$ of the total outline area.

The /BORDER Keyword

Read the information beginning on page 118 to learn more about panels and borders.

The optional /BORDER keyword is used to specify the longest dimension of any polygon on *layer1*. Shapes longer than this maximum dimension can be misclassified at panel boundaries. The /BORDER keyword is used to determine how all of the shapes in the design are processed in panels by the DRC. So do not use an arbitrarily large number since this will slow processing considerably. Too small a number may result in misclassified polygons. The default is /BORDER=0 which will prevent any polygons from being misclassified at the cost of executing the DRC in multiple passes through the data.

Whether or not /BORDER=0 is faster than /BORDER=*big_number* depends on your data and the other rules in your rule set. If other rules require a large border, try /BORDER=*big_number*. If you use CONNECT rules (required for electrical connection tests) or TOUCHING rules in your rule set, then the DRC must already process the data in multiple passes, and the default of /BORDER=0 will probably be faster.

Example: **RESULT = HOLE_AREA_FRACTION (A, 0.25, 1 /BORDER=50)**

This rule above will add a minimum border of 50 user units to the DRC panel processing. Shapes that have at least one side longer than 50 user units may be misclassified due to being sliced by a panel boundary during processing. However, this rule may execute more quickly than the default if the rest of the rule set can be executed in a single pass.

Counting Shapes as Errors

Refer to the OUTPUT LAYER rule to learn more about error layers.

The *result_layer* is not automatically considered an error layer, so shapes found by this rule will **not** automatically be counted as errors. You can process shapes on *result_layer* in the same manner as any output or scratch layer. However, if you define *result_layer* as an error layer (as shown on the next page), then shapes on the layer are counted as errors and will be reflected in the error count.

DRC Rules Syntax: HOLE_AREA_FRACTION

Example: **OUTPUT ERROR LAYER 41 A_WITH_BIG_HOLES**
 A_WITH_BIG_HOLES = HOLE_AREA_FRACTION (A, 0.25, 1)

IN_CELL

Classify shapes in certain cells

result_layer = *layer1* **IN_CELL** *cell_name*

The INCELL keyword of the INPUT LAYER rule **will not** include shapes in subcells of the specified cells.

This rule will classify shapes on a layer by whether or not they are contained in specific cells. It works in a similar manner to the IN_CELL parameter of the INPUT LAYER rule, however this rule processes the data differently in two ways:

layer1 can be any layer in the DRC database, not just an input layer

and

layer1 shapes in subcells of the specified cells **will be** included on *result_layer*.

Example:

A_IN_MYCELL = A INCELL MYCELL

The rule above will copy to the A_IN_MYCELL layer all shapes on layer A in cell MYCELL and it's subcells. Layer A in the database remains unchanged.

There are no optional NOT keywords in this rule. You can process the *result_layer* with an AND NOT rule if desired.

Example:

A_IN_CAP = A INCELL *PF
OTHER_A = A AND NOT A_IN_CAP

This pair of rules classifies shapes on layer A by whether or not they are contained in cells that end with the string "PF" or subcells of those cells.

For more examples of *cell_name* specifications, including using wildcards, see page 219.

INCLUDE

Allow rules file nesting

INCLUDE [*dir_path*]*file_name*

This rule allows you to nest rules files. An INCLUDE rule in one rules file will result in another rules file being inserted at that point. The *file_name* parameter is used to specify the name of the file. The file extension (if any) must be included in *file_name*.

Example: **INCLUDE MOSCONST.RUL**

This rule will cause the text in the file MOSCONST.RUL to be added to the current rules file at the point where the INCLUDE rule is found. Since no *dir_path* parameter is used, the file MOSCONST.RUL must exist in the current directory.

You may optionally supply a directory path with the file name. You should fully qualify the directory. You can place quotes around the file name if the DRC rules preprocessor may have any problems parsing the file name. This will be the case if the file name contains blanks as in the example below.

Example: **INCLUDE "C:\ICED\MOS 123.RUL"**

You **cannot** use the INCLUDE rule in the middle of another rule. You may nest rules files up to 10 deep with the INCLUDE rule.

INPUT LAYER

Define input layers

INPUT LAYER *iced_layer_number_1* ...
... [+ *iced_layer_number_2* [... + *iced_layer_number_5*]] ...
... [[NOT]⁹ INCELL *cell_name*]¹⁰ ...
... *drc_layer_name* ...
... [NOT *drc_not_incell_layer_name*]⁹

See an overview
of layer
definition rules
on page 55.

All layers in the input data that will be used in your DRC rule set must be defined in an INPUT LAYER rule. The only required parameters for the INPUT LAYER rule are *iced_layer_number_1* and *drc_layer_name*. The *iced_layer_number* parameters correspond to the layer numbers in the ICED™ cell. (The layer names used in the ICED™ cell are ignored by the DRC.) A specific *iced_layer_number* can be referred to only once in your set of INPUT LAYER rules.

You can specify
a layer number
at run time with
the LAYERS
option on the
DRC command
line. See page
346.

The shapes on DRC layers created with the INPUT LAYER rule **cannot** be modified by other rules. If you need to modify an input layer, you can use the assignment rule (page 187) to copy the layer to a scratch or output layer. Use a MODIFY LAYER rule instead of INPUT LAYER to define a layer used as both an input layer and an output layer.

The *drc_layer_name* is the label used to specify the layer in other DRC rules. The name does not need to be identical to the layer name in the ICED™ cell. A specific *drc_layer_name* can appear only once in your set of INPUT LAYER rules.

⁹ Only one optional NOT keyword is allowed in a single rule.

¹⁰ More than 1 INCELL *cell_name drc_layer_name* pair is allowed. See page 220.

DRC Rules Syntax: INPUT LAYER

Example: **INPUT LAYER 2 M1**

When this rule is used, all components on layer 2 in the ICED™ main cell, and all subcells, will be copied to layer M1 in the DRC database. Use the name M1 to refer to this layer in succeeding DRC rules.

If you want to combine shapes on several ICED™ layers into one DRC layer, specify several *iced_layer_number* parameters separated with plus signs ('+'). You can combine up to five ICED™ layers into one DRC layer.

Example: **INPUT LAYER 2 + 12 + 22 M1**

This rule will combine the ICED™ layers 2, 12, and 22 into the DRC layer M1.

When you need to define many input layers, you can list several input layers in one INPUT LAYER rule. Separate the layers with semicolons(';').

Example: **INPUT LAYER 1 A; 2 B ; 3 C**

When an input layer definition is split over more than one line, you can surround the layer definition with curly braces { }. If you type one layer definition on each line, semicolons are not required.

Example: **INPUT LAYER {
 1 A
 2 B
 3 C
 }**

All input layers are checked for bad polygons by default. For this reason, it is a good idea to define all mask layers as input layers, even if they are not verified by any rules.

Note that the '&' continuation character is **not** required to split this example across several lines.

Restricting Input Layers by Subcell

The INCELL options, [[NOT] INCELL *cell_name*] and [NOT *drc_not_incell_layer_name*], are used to classify components on an input layer by whether or not they are stored in specific subcells.

Example: **INPUT LAYER 2 INCELL INDUCTOR_CELL INDUCTOR_M1**

The IN_CELL rule, which also classifies shapes by cell, **will** include shapes in subcells of the specified cells.

This rule will copy all components on layer 2 contained in instances of cell INDUCTOR_CELL (but **not** its subcells) to DRC layer INDUCTOR_M1.

The *cell_name* in the INCELL parameter can contain wildcard characters (*). A vertical bar, | can be used as well to indicate a list of valid cell names. More than one | delimiter can be used. Do not use any blanks when entering the *cell_name* parameter.

Example: **INPUT LAYER 2 INCELL IND*|*NH IND_M1**

This input layer specification will copy to layer IND_M1 all components on layer 2 contained in cells which begin with the string "IND", or which end in the string "NH".

You can refer to the main cell (the highest-level cell) of your input data with the special name "@MAIN". This allows you to use the same rule set for different designs that use different names for the main cell.

Example: **INPUT LAYER 1 INCELL @MAIN LAY1_MAIN**

In this example, shapes on layer 1 that are contained in the main cell, but not its subcells, will be copied to DRC layer LAY1_MAIN. The actual name of the main cell is irrelevant.

The NOT keywords are used to indicate that the layer contains only shapes on layer *iced_layer_number* which are **not** contained in the specified cells. Only one NOT keyword is allowed.

You would use the first optional NOT keyword to restrict the new layer to shapes that are **not** contained in the cell(s) indicated after the INCELL keyword.

DRC Rules Syntax: INPUT LAYER

Example: **INPUT LAYER 3 NOT INCELL *PF* DIFF**

This rule will create the DIFF layer with all shapes on layer number 3 that are not contained in cells that contain the string "PF".

You would use the second optional NOT keyword when you need to classify shapes on the indicated layer number into different DRC layers: one DRC layer for shapes in the cell(s), and another DRC layer for those shapes which are **not** in the cell(s). When the second optional NOT keyword is used, *drc_layer_name* is restricted to shapes which **are** contained in the cells indicated after the INCELL keyword, and layer *drc_not_incell_layer_name* will contain shapes which are **not** in the indicated cells.

Example: **INPUT LAYER 2 INCELL INDUCTOR_CELL IND_M1**
 INPUT LAYER 2 NOT INCELL INDUCTOR_CELL M1 !Error

These 2 statements together would cause a compiler error since each *iced_layer_number* can occur in only one INPUT LAYER rule. You can achieve the desired result with the following single statement:

Example: **INPUT LAYER 2 INCELL INDUCTOR_CELL IND_M1 NOT M1**

When you need to classify a single layer number into several different DRC layers, you can use more than one INCELL specification in a single rule. You can specify up to 50 INCELL *cell_name drc_layer_name* pairs in a single INPUT LAYER rule.

Example: **INPUT LAYER 1 INCELL CAP12PF CAP12_DIFF &**
 INCELL CAP104PF CAP104_DIFF &
 INCELL CAP1200PF CAP1200_DIFF &
 NOT DIFF

The example above separates layer number 1 in the input data into 4 different DRC layers. Shapes on layer 1 in cells with the name CAP12PF will go into DRC layer CAP12_DIFF, etc. Shapes on layer 1 that are not contained in cells with the names CAP12PF, CAP104PF, or CAP1200PF will be placed in DRC layer DIFF.

See page 172 to get more details on using '&' to split a rule over several lines.

Note that the '&' continuation character is required to split this example above over several lines.

Restricting Input Layers by Subcell Boundaries

Layer 0 in an ICED™ cell is used to store subcell bounding boxes. Ordinary shapes are never stored on that layer. In an INPUT LAYER statement, layer 0 can be used to store a rectangle that covers a subcell. This may be useful in some types of layer processing.

Example:

```
INPUT LAYER  0 INCELL INDUCTOR_CELL IND_MASK
INPUT LAYER  2 M1_IN
M1 =        M1_IN AND NOT IND_MASK
IND_M1 =    M1_IN AND      IND_MASK
```

When this set of statements is used to classify layer M1 instead of the example on page 220, you must be careful with an important side effect. In this example, the processing on M1 is performed after the cell is flattened hierarchically. Shapes on M1 in subcells of INDUCTOR_CELL (or in the main cell, or any other cell) which happen to be located within the bounding box of INDUCTOR_CELL will also be classified as IND_M1. This can be desirable or not, depending on how your design is organized.

Classify rectangles by size

¹¹ Only one optional NOT keyword is allowed in a single rule.

Example: **B = IS_BOX (A, (10, 5))**

This rule will collect on layer B all rectangles on layer A which are 10 units wide in the x-direction and 5 units high in the y-direction. The commas and parentheses are required.

Note that orientation is important. To collect non-square rectangles which may be in either orientation you **must** specify two sizes. The DRC recognizes dimensions in the x and y directions separately due to the fact that in some technologies (e.g. the Gallium Arsenide technology) the orientation is important.

Example: **B = IS_BOX (A, (10,5), (5,10))**

This rule will collect on layer B all rectangles on layer A which are 10 units wide by 5 units high in either orientation.

Example: **B = ISBOX (A, (10:12, 5:7))**

Here, layer B will consist of all rectangles on layer A which are from 10 to 12 units wide in the x-direction and from 5 to 7 units high in the y-direction.

(Note that the underscore in the IS_BOX keyword is optional. The underscore character is simply ignored when it is present. This is true of all keywords.)

Example: **B = IS_BOX (A, (10:12, 6.4))**

This rule will collect on layer B all rectangles on layer A which are from 10 to 12 units wide in the x-direction and exactly 6.4 units high in the y-direction.

The optional NOT keywords are used to restrict the output layer to all shapes that do not meet the size criteria. Only one optional NOT keyword is allowed.

Example: **C = NOT IS_BOX (A, (10, 2), (11, 3))**

This above rule will collect on layer C all shapes on layer A which are **not** rectangles 10 units wide and 2 high or 11 units wide and 3 high.

DRC Rules Syntax: IS_BOX

When typing this rule, you may start a new line between sizes. You cannot split a single *size*n parameter between lines. (The final optional NOT keyword must be on the same line as the closing parentheses.)

Example: **B = IS_BOX (A,**
 (10, 2), (2, 10),
 (5, 1), (1, 5)) NOT = C

The rule above will collect on layer B all shapes on layer A which are rectangles 10 units wide and 2 high or 5 units wide and 1 high in either orientation. Layer C will consist of all other shapes on layer A, including all non-rectangular shapes.

IS_CIRCLE

Classify polygons with circular shape

```
result_layer = [NOT]12 IS_CIRCLE ( layer1, ...  
    ... R= min_radius: max_radius[,] ...  
    ... N= min_sides [ : max_sides ][,] ...  
    ... EPS=tolerance[,] ...  
    ... [POLY_INSIDE] [POLY_OUTSIDE][,] ...  
    ... ) [NOT = result_layer_2]12
```

This rule is used to classify polygons on *layer1* based on whether or not they approximate circles that meet certain criteria.

(Remember that all shapes on the same layer are merged by the DRC. Circular polygons that touch another shape on the same layer will be merged during preprocessing. When a circular shape is merged with touching shapes, the resulting shape may no longer be circular.)

The specifications for minimum radius, maximum radius, minimum number of sides, and the EPS tolerance are all required.

Learn more
about the
resolution grid
on page 79.

The EPS tolerance is a spacing tolerance that allows the vertices of a polygon to be slightly displaced from where they would be in a perfect circle. This tolerance is required to find circular polygons since their vertex coordinates always vary from ideal coordinates due to the resolution grid.

Example: **B = IS_CIRCLE (A, R = 1 : 5, N = 8, EPS = 0.01)**

This rule will collect on layer B all polygons on layer A that approximate circles with radii in the range 1.0 to 5.0 user units. The polygons must have at least 8 sides. This EPS value is typical for polygons in this size range.

¹² Only one optional NOT keyword is allowed in a single rule.

Specifying Radii

It is a good idea to broaden the size range slightly to improve your chances at finding all polygons that are close to your size criteria. The true radius of a circular polygon is likely to vary slightly from the equivalent ideal circle.

Example: **B = IS_CIRCLE (A, R = 4.99 : 5.01, N = 8, EPS = 0.01)**

This rule will find all polygons that approximate circles with a radius of 5 user units. Restricting the size criteria to a single number (e.g. R = 5.0 : 5.0) is likely to prevent the DRC from finding any polygons.

Specifying the Number of Sides

If you want to refine the search to find only polygons that are more circular than octagons, increase the minimum number of sides indicated after the 'N' keyword.

Example: **B = IS_CIRCLE (A, R = 4.99 : 5.01, N = 16, EPS = 0.01)**

This rule will find all polygons that approximate circles with a radius of 5 user units and have at least 16 sides.

The minimum and optional maximum number of sides must be expressed as positive integers. When the maximum number of sides is not provided, it defaults to a very large number (over 2 billion).

Example: **B = IS_CIRCLE (A, R = 4.99 : 5.01, N = 3:5, EPS = 0.01)**
 C = IS_CIRCLE (A, R = 4.99 : 5.01, N = 6, EPS = 0.01)

This pair of rules will find circular polygons with a radius of 5 user units on the A layer. Layer B will include only polygons with from 3 to 5 sides (i.e. equilateral triangles, squares, and pentagons). Layer C will include only polygons that have at least 6 sides.

The Optional POLY_INSIDE and POLY_OUTSIDE Keywords

These mutually exclusive keywords determine which mode is used to determine how the ideal circle relates to the approximating polygon. The POLY_OUTSIDE keyword is used by default when the POLY_INSIDE keyword is not used. When the POLY_OUTSIDE mode is used, the polygon is understood to be drawn outside of the ideal circle. This is the method used by the ICED™ layout editor when it creates a circular polygon.

The POLY_INSIDE keyword will change the radii criteria to search for circular polygons where the polygon is smaller than the ideal circle.

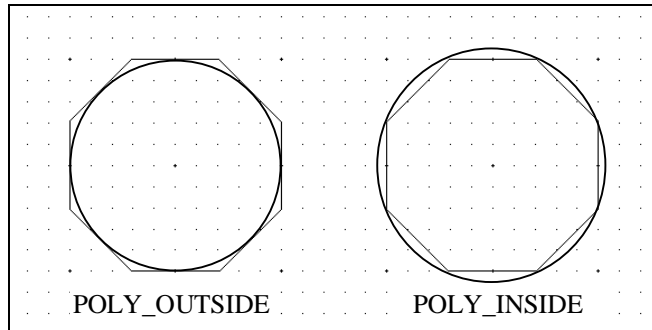


Figure 142

Example: **B = IS_CIRCLE (A, R = 3.99 : 5.01, N = 16, EPS = 0.01, POLY_INSIDE)**

The rule above will search for circular polygons adjusting the radius criteria so that polygons drawn inside ideal circles of radii from 4 to 5 will be copied to the B layer. The polygons must have at least 16 sides.

Using the NOT Keywords

The optional NOT keywords are used to collect all polygons on *layer1* that do not meet the circle criteria. Only one NOT keyword is allowed in a single rule.

DRC Rules Syntax: IS_CIRCLE

Example: **B = IS_CIRCLE (A, R = 4.99 : 5.01, N = 8, EPS = 0.01) NOT = C**

This rule will copy to layer B all shapes on layer A that meet the circle criteria. All other shapes on layer A (including all non-circular shapes) will be copied to layer C.

If you need to collect only shapes that do not meet the circle criteria, use the first NOT before the IS_CIRCLE keyword as in the example below.

Example: **C = NOT IS_CIRCLE (A, R = 4.99 : 5.01, N = 8, EPS = 0.01)**

To improve readability, you can split this rule across several lines. Begin a new line after any of the commas. The only restriction is that if the second NOT is used on the last line by itself, the closing parentheses must be included in that last line.

Example: **B = IS_CIRCLE (A,
 R = 4.99 : 5.01
 N = 8
 EPS = 0.01
) NOT = C**

Note that the commas are optional and can be omitted (except for the first comma after the layer name).

Counting Shapes as Errors

Refer to the
OUTPUT
LAYER rule to
learn more
about error
layers.

The *result_layer* is not automatically considered an error layer, so shapes found by this rule will **not** automatically be counted as errors. You can process shapes on *result_layer* in the same manner as any output or scratch layer.

If you define *result_layer* as an error layer (as shown in the next example), then shapes on the layer are counted as errors and will be reflected in the error count.

Example:

OUTPUT ERROR LAYER 53 CIRCLE_ERR

CIRCLE_ERR = IS_CIRCLE (A, R = .1 : 999, N = 6, EPS = 0.01)

A rule similar the one above will find all circular components. All of the shapes copied to the CIRCLE_ERR layer will be counted as errors since the *result_layer* is defined as an error layer.

ISLANDS

Find Holes

result_layer = **ISLANDS** (*layer1*)

Also see the
HOLE_AREA_
FRACTION
rule to find
polygons with
holes.

This rule is used to find holes or unconnected polygons on a specific layer. All shapes on *layer1* that are not connected to the upper left polygon on *layer1* will be copied to *result_layer*. By connected, we do not mean electrical connections through use of the CONNECT rule. For this rule, connected means shapes on one layer which touch other shapes on the same layer.

The *result_layer* is not automatically an error layer. Shapes generated on the *result_layer* will not count as errors unless you add the ERROR keyword to the OUTPUT LAYER rule that defines the layer.

To find holes in a layer, you use this rule to find islands in the inverse of the layer.

Example:

NOT_A = NOT A
B = ISLANDS (NOT_A)

This pair of rules will result in polygons on layer B created for all holes in layer A. Note that the parentheses are required in the ISLANDS rule.

MAX_ANGLE

Find sharp points in notches

error_layer = MAX_ANGLE (*layer1*, *angle*)

See the MIN_ANGLE rule to find acute angle protrusions.

This rule is used to find acute angle notches in polygons. The angle measured is the interior angle.

Look at Figure 143. The angle of the notch is 36.9° . The interior angle of the polygon is 323.1° .

Example:

ERR1=MAXANGLE (A, 315)

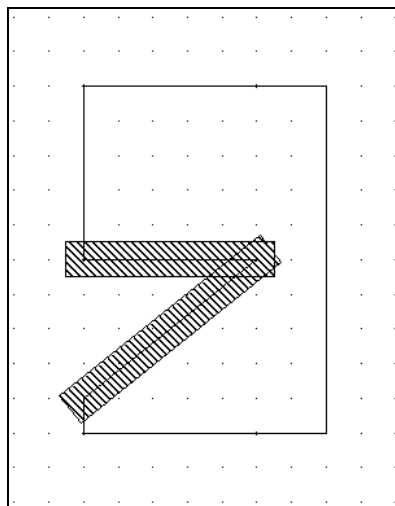


Figure 144: Error wires marking notch.

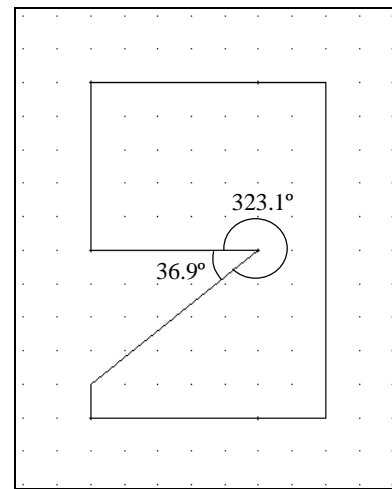


Figure 143: Polygon with acute angle notch.

The rule above will find all polygons with interior angles greater than 315° . This means notches with angles less than:

$$360^\circ - 315^\circ = 45^\circ$$

If this rule is executed on the polygon shown in Figure 143, the angle will be marked with error wires on layer B, since it is less than 45° . The error wires will look similar to Figure 144.

You usually want the *angle* parameter to be at least 270°. If *angle* is less than 270°, then all 90° bends like the one shown in Figure 145 will be marked as errors. To find only acute angle notches, use values for *angle* in the following range:

$$360^\circ > \text{angle} > 270^\circ$$

This will find notches with angles less than:

$$0^\circ < \text{notch_angle} < 90^\circ$$

To be found by this rule, notches must be formed from two connected sides that meet at an angle. If the notch is blunted by another line segment, this rule will not find it. Refer to Figure 146. Since the angular notch is blunted by the vertical segment, neither angle is greater than 315°. See the MIN_NOTCH rule to find notches like these.

If you want to find acute angle protrusions on a specific layer, see the MIN_ANGLE rule. If you want to find all acute angles (protrusions and notches) on all output layers, see the information in the WARN_ACUTE rule description.

All shapes created on *error_layer* by this rule are counted as errors. They will automatically be included in the DRC error count.

All acute angles on output layers are marked with warnings on layer 99 by default.

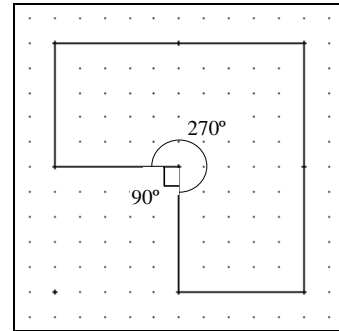


Figure 145: Polygon with 270° interior angle.

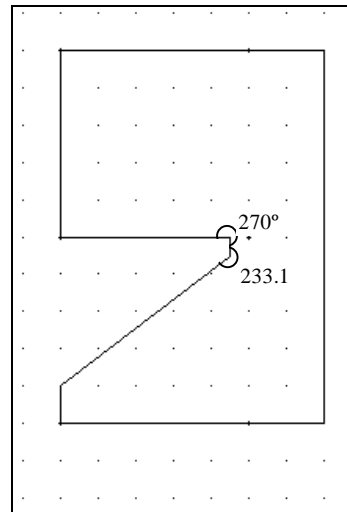


Figure 146: Notch that will not be marked by the MAX_ANGLE(A,315) rule.

MAX_COUNT *Change maximum number of errors found before warning*

MAX_COUNT = *error_count*

Add the STOP_ON_-MAX_COUNT rule to the rule set if you prefer to have the DRC halt when the error count reaches the maximum.

The DRC will warn you by posting a message on the screen when a maximum error count is reached. When you do not include the MAX_COUNT rule in your rule set, the default maximum error count is 1000. Use this rule to change this maximum error count.

To understand why the DRC warns you when a maximum error count is reached, imagine a chip with 10,000 copies of a cell. If a small change to this cell causes a single error, there will be at least 10,000 error marks created for what you would consider a single error. Other error marks will be easily overlooked. The error would be caught just as well if you halted the DRC with the <Esc> key after finding the first 1000 errors, and the run time and output files would be much smaller. It is much more efficient to find and fix the single error in a shorter run, and then other errors will be easily seen in your next run.

Example: **MAX_COUNT = 3000**

This rule will cause the DRC to post a warning message similar to the following as soon as it has found 3000 errors.

```
*****WARNING*****WARNING*****WARNING*****
****Error count = nnnn*****
You may stop this run by pressing <Esc>. There will be a delay. Pressing
<Ctrl><C> or <Ctrl><Alt><Del> will lose data already generated.
```

You can ignore the message and allow the DRC to run to completion. The message will be updated from time to time with the current raw error count.

If you want to stop the run, you should press the <Esc> key. The current pass will be completed, the log file will be generated, and any scratch file(s) will be deleted before the DRC comes to a halt. This may take a few minutes. If you

DRC Rules Syntax: MAX_COUNT

press <Ctrl><C>, the DRC will halt immediately, but files will not be closed properly and the scratch file(s) will need to be deleted manually.

A MAX_COUNT larger than the default will allow you to find many more errors than the default without re-executing the DRC.

However, do not set *error_count* to a large number like 3000 for an early DRC run using a new rule set. If many errors are caused by mistakes in your rule set, it will take you much longer to realize this, when a shorter run would have been adequate to debug your rule set.

The DRC keeps track of a raw error count as it encounters errors during the run. In many cases, several errors get added to the raw error count are combined during later processing into a single error. Therefore, if an error count warning is generated by the DRC, but you allow the DRC to keep running, you may see that the final error count is lower than the raw running total posted in the warning message.

MAX_SPACING

Classify shapes by distance

```

result_layer = [NOT]13 MAX_SPACING ( from_layer [/IN] [/OUT] [/CAP=angle1], ...
... to_layer [/IN] [/OUT] [/CAP=angle2], ...
... distance ...
... [/~]CROSS] ...
... [/~]PERP] ...
... [/~]T] ...
... [/~]OVER] ...
... [/~]END] ...
... [/~]INTER] ...
... [/~]CONN] ...
... ) [NOT = result_layer_2]13

```

This rule will classify shapes on the *from_layer* by whether or not they are **more than** *distance* away from sides of shapes on the *to_layer*. Set *distance* to a positive real number of user units.

Example: **RESULT = MAX_SPACING (A, B, 2)**

The shapes found by this rule are not considered errors. If you want to mark them as errors, see the example at the end of this description.

This rule will copy to layer RESULT all shapes on layer A that have all vertices more than 2 user units away from sides of shapes on layer B.

One quirk of this rule is that **overlapping shapes can be classified as being more than *distance* away even though they share a common area.** A shape on the *from_layer* can be copied to the *result_layer* as long as all vertices of the shape are further than *distance* away from sides of shapes on the *to_layer*.

¹³ Only one optional NOT keyword is allowed in a single rule.

Look at the 3 small rectangles in Figure 147 that all represent shapes on layer A. If the rule above is executed on these shapes, both shapes 1 and 2 will be copied to RESULT.

Shape 1 will be copied to RESULT since all of its vertices are more than 2 user units away from the B shape.

All of the vertices of shape 2 are also more than 2 units away from a side of the shape on layer B. So shape 2 is also copied to RESULT, even though it is covered by the shape on layer B.

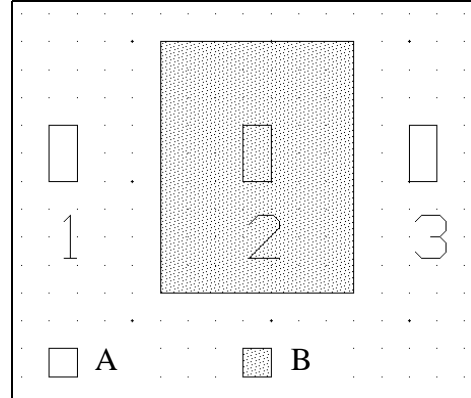


Figure 147: Both layer A shapes 1 and 2 are more than 2 units away from the sides of the layer B shape.

Since shape 3 has vertices that are **exactly** 2 user units away, it is **not** copied to layer RESULT.

If overlapping shapes are special cases, use Boolean rules to modify the shapes on the *from_layer*, the *to_layer*, or the *result_layer*. For example, if you want to remove overlapping shapes like shape 2 in the example above, use rules similar to the following:

Example:

A_NOT_B = A AND NOT B
RESULT = MAX_SPACING (A_NOT_B, B, 2)

When this pair of rules is used on the shapes in Figure 147, only shape 1 will be copied to the RESULT layer.

If you need to find shapes on a single layer that are more than *distance* apart, the *to_layer* can be the same as the *from_layer* as in the following rule:

Example:

RESULT = MAXSPACING (A, A, 1.5)

All layer A shapes that are more than 1.5 user units away from other all other layer A shapes are copied to RESULT. Note that the underscore ('_') is optional in the MAX_SPACING keyword (and all DRC keywords).

Using the NOT Keywords

Use either NOT keyword to copy all shapes on the *from_layer* that do **not** meet the spacing criteria to a result layer. Only one NOT keyword is allowed in a single rule.

Example: **RESULT = MAX_SPACING (A, A, 1.5) NOT=RESULT2**

Adding "NOT = RESULT2" to the previous example will copy all layer A shapes that are 1.5 user units or less away from another layer A shape to the RESULT2 layer.

If you need to collect only shapes that do not meet the spacing criteria, use the first NOT before the MAX_SPACING keyword as in the example below:

Example: **RESULT2 = NOT MAX_SPACING (A, A, 1.5)**

Using this MAX_SPACING rule with a NOT keyword is very similar to using the equivalent MIN_SPACING rule below:

ERR = MIN_SPACING (A, A, 1.5)

The three differences are:

- The *result_layer* shapes are not counted as errors by the MAX_SPACING rule.
- The entire shape on the *from_layer* is copied to the result layer by the MAX_SPACING rule so you can manipulate it just like any other shape. MIN_SPACING produces error wires that cannot be used for other processing.
- Shapes that are exactly distance apart are treated differently.
 - MIN_SPACING finds shapes that are less than *distance*.
 - MAX_SPACING finds shapes that are more than *distance*.
 - NOT MAX_SPACING finds shapes that are less than or equal to *distance*.

Directional Spacing Checks, End Caps, and Orientation Options

See the MIN_SPACING rule and Spacing Verification beginning on page 84 to learn more about how the DRC performs spacing verification.

The /IN, /OUT, and /CAP options for the input layers and the orientation options /CROSS, /PERP, /T, /OVER, /END, and /INTER are very rarely used. They usually increase the number of shapes that are copied to the *result_layer*. For example, when the /IN or /OUT options are added to the *from_layer* or *to_layer* in a MAX_SPACING rule, this allows shapes that are closer than *distance* to be added to the *result_layer* by restricting the spacing criteria to those shapes found by looking toward the inside or toward the outside of shapes on indicated layer.

These options are included for completeness since they are included in the MIN_SPACING rule. The algorithms for the MAX_SPACING rule are based on those used for the MIN_SPACING rule. Basically, shapes that have no sides that violate the equivalent MIN_SPACING rule are copied to *result_layer*. For example, if we restrict the equivalent MIN_SPACING rule with /IN or /OUT, less shapes fail the MIN_SPACING test. So more shapes pass the equivalent MAX_SPACING rule.

There is one important thing to keep in mind if you are using these MIN_SPACING options in a MAX_SPACING rule. Shapes on the *from_layer* that are exactly *distance* away from a shape on the *to_layer* at their closest point are not copied to the *result_layer* even though they would not violate the equivalent MIN_SPACING rule.

Read the relevant section in the MIN_SPACING rule if you need to understand these options. Unless you are writing a simple MAX_SPACING rule, you should be familiar how the equivalent MIN_SPACING rule operates.

One final word on this subject for those who understand MIN_SPACING well enough to be dangerous: none of these options will prevent overlapping polygons from being copied to *result_layer*. Use the Boolean method shown on page 236 to prevent overlapping shapes from being copied to the *result_layer*.

Electrical Connections

When the rules compiler or DRC lists a MAX_SPACING rule with the default connection restriction "/+~CONN", this means both connected and unconnected pairs of shapes will be checked.

Add the /CONN option to the MAX_SPACING rule if you want to copy to the *result_layer* all shapes on the *from_layer* that are more than *distance* away from shapes on the *to_layer* to which they are electrically connected. The /~CONN option will copy shapes that are more than distance away from shapes that are not electrically connected. The default is to check both connected and unconnected pairs of shapes.

For MAX_SPACING rules to accurately recognize what shapes are electrically connected, you must define how electrical connections are made. You use CONNECT and STAMP rules to define electrical connectivity. See page 110 for a complete explanation.

Example:

CONNECT A B
RESULT = MAXSPACING (A, A, 2.0 /CONN)

The CONNECT rule above indicates that touching shapes on layers A and B are electrically connected. Since the MAX_SPACING rule includes the /CONN option, any given shape on layer A will be copied to RESULT if it is more than 2 user units away from all other shapes on layer A to which it is electrically connected.

Look at Figure 148. All squares represent shapes on layer A. Shapes 1, 2, 3, and 4 are all electrically connected by the vertical rectangle on layer B. Shapes 1 and 2 are not copied to RESULT since they are electrically connected and closer than 2 units. Shape 3 is also not copied to RESULT since it is exactly 2 units away from an electrically connected layer A shape. All other shapes will be copied to RESULT since they are more than 2 user units away from other electrically connected layer A shapes.

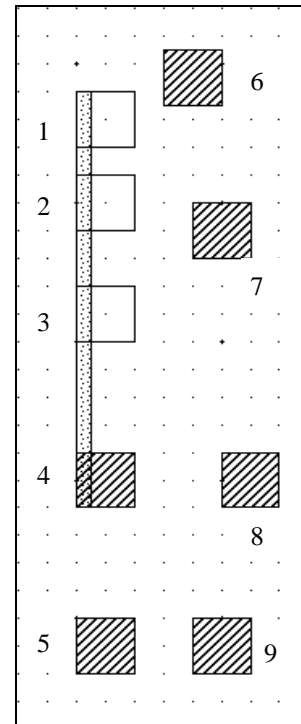


Figure 148

DRC Rules Syntax: MAX_SPACING

For the next example, suppose that you need to find all layer A shapes that are within 2 user units of another shape on layer A that is on a different electrical net. Use the `/~CONN` option and collect only shapes that fail to meet the `MAX_SPACING` criteria.

Example: `RESULT2 = NOT MAXSPACING (A, A, 2.0 /~CONN)`

Now shapes on layer A that are 2 units or closer to shapes that are not electrically connected are copied to `RESULT2`. Shapes 1 and 2 are copied because they are too close to shapes 6 and 7 respectively. Note that shapes that are **exactly** 2 units away from shapes that are not electrically connected **are** copied to `RESULT2`. When the `NOT` keyword is used, shapes that are exactly distance away or closer are collected on the result layer.

See page 129 for more information on the `QUICKPASS` option.

NOTE: The restrictions imposed by the `/CONN` or `/~CONN` options will be ignored when you specify the `QUICK_PASS` option on the DRC command line. **You should use the `SLOW` command line option to enable the `/CONN` or `/~CONN` options.**

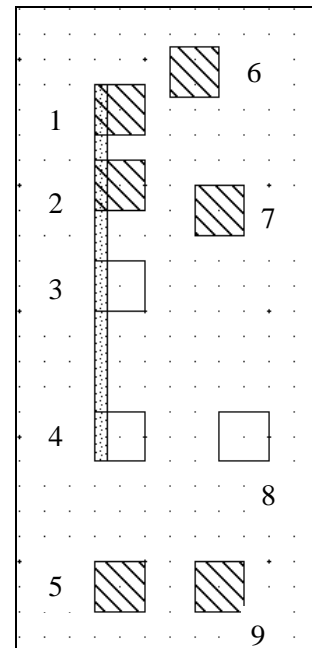


Figure 149:
MAXSPACING (A, A,
2.0 /~CONN)

Counting Shapes as Errors

Refer to the `OUTPUT LAYER` rule to learn more about error layers.

The *result_layer* is not automatically considered an error layer, so shapes found by this rule will **not** automatically be counted as errors. You can process shapes on *result_layer* in the same manner as any output or scratch layer. If you define *result_layer* as an error layer (as shown below), then shapes on the layer are counted as errors and will be reflected in the error count.

Example: **OUTPUT ERROR LAYER 91 A_TOO_FAR_AWAY**
 A_TOO_FAR_AWAY = MAXSPACING (A, A, 1.5)

All layer A shapes that are more than 1.5 user units away from other all other layer A shapes are copied to layer A_TOO_FAR_AWAY. Since this is defined as an error layer, all shapes on the layer are added to the error count.

MIN_ANGLE

Find sharp points

error_layer = **MIN_ANGLE** (*layer1*, *angle*)

This rule is used to find acute angles in polygons. The angle measured is the interior angle.

Example:

ERR1=MIN_ANGLE (A, 45)

See the WARN_ACUTE rule to learn how the DRC will find acute angles on **all** output layers.

This rule will find all polygons with angles less than 45°.

If this rule is executed on the polygon shown in Figure 150, the angle will be marked with error wires on layer B, since it is less than 45°. The error wires will look similar to Figure 151.

See the MAX_ANGLE rule to find acute angle **notches** on polygons.

You usually want to restrict *angle* to be in the following range:

$$0^\circ < \text{angle} < 90^\circ$$

If you specify *angle* to be greater than 90°, all right angle corners of all polygons on *layer1* will be marked.

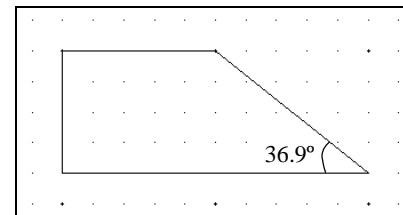


Figure 150: Polygon with acute angle.

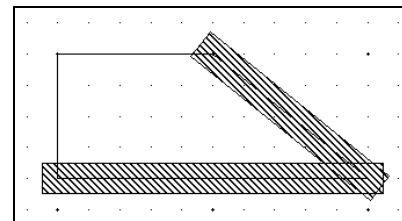


Figure 151: Error wires marking acute angle.

MIN_AREA

Find small shapes

error_layer = **MIN_AREA** (*layer1*, *area*, /BORDER= [+] *max_size*)

This rule will copy to the *error_layer* (and count as errors) all polygons on *layer1* with an area less than *area*.

See page 118 to learn more about panels and borders.

The **required** /BORDER keyword is used to modify the panel border. It is used to avoid false errors for shapes that fail to pass the test because they are formed by touching shapes that travel across panel boundaries.

The SHOW command in the ICED™ layout editor will report the area of selected shapes.

There are three ways you can specify the border:

/BORDER=*max_size* sets the panel border to a minimum of *max_size*.

/BORDER=+*max_size*¹⁴ adds *max_size* to the border value required by other rules.

/BORDER=0 forces the DRC to use multiple passes which prevents false error messages entirely.

See page 66 for an example of using this rule to filter small shapes without counting them as errors.

In no case will any of these choices prevent real violations from being found. However, a small border can fail to prevent false errors, and a large one can result in longer run times for the entire DRC run.

Generally, the /BORDER=0 option is the most efficient. If your rule set contains other verification rules for *layer1*, and the rules compiler has generated a "CONNECT *layer1*" rule (look in the rule compiler log file), then this option will not result in a longer run time. The DRC will already execute in multiple passes.

¹⁴ This special syntax to modify the border is unique to the MIN_AREA rule.

DRC Rules Syntax: MIN_AREA

Add the SHOW-
_BORDER
option to the
DRC command
line to see how
the panel border
is calculated in
the log file.

However, if you have a rule set that contains no other rules that force the compiler to generate the CONNECT rule above, using the /BORDER=*max_size* option may be faster than the /BORDER=0 option.

We suggest using a small value for *max_size* unless you are seeing too many false errors. If many false errors are a problem, a good value for *max_size* is:

$$\frac{area}{minimum_width}$$

where *area* is the minimum area set in the rule, and *minimum_width* is the minimum dimension allowed for *layer1* in your technology.

Look at the shapes in Figure 152. The upper shape has an area of 8. If the touching shapes at the bottom are merged, they have a combined area of exactly 10. The merged shapes should pass the minimum area check.

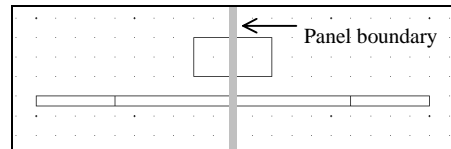


Figure 152: Shapes on layer A.

Example:

PANELX = 5
B=MIN_AREA (A, 10 /BORDER=1)

When these rules are run on the shapes in Figure 152, the indicated panel boundary may prevent all 4 shapes on the bottom from being considered as the same shape. In this case the DRC will mark false errors for these shapes as shown in Figure 153. (The shapes on layer B are cut at the panel border. The log file will include all 6 shapes on layer B in the count of errors.)

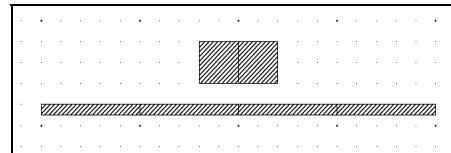


Figure 153: All shapes marked with errors on layer B.

Example:

PANELX = 5
B=MIN_AREA (A, 10 /BORDER=20)

When we increase the border to 20, only the upper shape will be copied to layer B. All of the lower shapes are now considered one shape with an area of exactly 10, so they pass this minimum area rule.

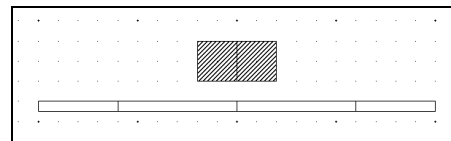


Figure 154: Only upper shape is marked when larger border is used.

MIN_FILL**Verify layer coverage of design area**

```
error_layer = MIN_FILL { layer1, min_fraction, ...  
    ... [MARGIN=mdistance]...  
    ... [ LEFT_MARGIN=ldistance[,] ...  
    ... RIGHT_MARGIN=rdistance [,] ...  
    ... TOP_MARGIN=tdistance [,] ...  
    ... BOTTOM_MARGIN=bdistance ]...  
    }
```

The MIN_FILL rule checks that a minimum fraction of the total design area is covered by shapes on *layer1*. The total area of all shapes on *layer1* within the design area is calculated and divided by that design area. If the resulting number is less than *min_fraction*, then the DRC will post an error message in the log file and create a text component on *error_layer*. The *error_layer* text component is created at the lower left corner of the design. A larger warning message text component is created at the top of the design on *error_layer* to alert you to the specific warning text in the lower left corner.

To pass this rule, the following equation must be true:

$$\frac{\text{Total area of shapes on } layer1}{\text{Total area of design}} \geq min_fraction$$

The bounding box of the design sets the default total area of the design. **The bounding box is the smallest rectangle, square with the axes, which encloses all shapes on all layers in the design.** (This includes layers not used in your rule set.)

Example: **ERR = MIN_FILL {C .5 }**

The rule above will verify that shapes on the C layer cover at least ½ of the area of the entire design. The bounding box of all shapes on all layers determines the boundary of the entire design. If layer C covers at least half of this area, no errors are generated. If the total area of all shapes on layer C is less than half of the design area, then a text message stating this, along with exact recommendations to fix this problem, is created in a text component in the output command file at the lower left corner of the design on layer ERR. This will be counted as an error in the error count.

If you want to define the total design area as an area slightly larger than the bounding box of the entire design, you can use either one of the following options:

- MARGIN=*mdistance*
- All four of the boundary options:
 LEFT_MARGIN=*ldistance*,
 RIGHT_MARGIN=*rdistance*,
 TOP_MARGIN=*tdistance*,
 BOTTOM_MARGIN=*bdistance*,

If you want to add the same margin distance to all sides of the bounding box of the design to specify the total design area, add the MARGIN=*mdistance* option to the rule. *mdistance* will be subtracted from the left and bottom boundary coordinates and added to the right and top coordinates. Specify *mdistance* as a positive real number of user units.

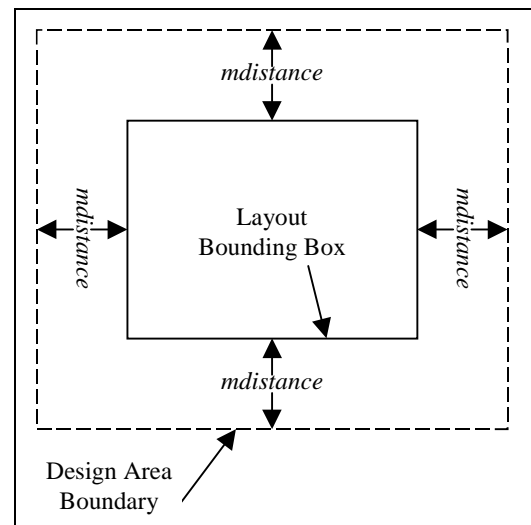


Figure 155: Using MARGIN=*mdistance*

Example: **ERR = MIN_FILL {C .25 MARGIN=50}**

The rule above will check that shapes on layer C cover at least $\frac{1}{4}$ of the design area. The design area in this case is considered to be 100 user units wider and 100 user units taller than the bounding box of the design data.

If you want to add different margin distances to all sides of the bounding box of the design to specify the total design area, add **all four** of the
 LEFT_MARGIN=*ldistance*,
 RIGHT_MARGIN=*rdistance*,
 TOP_MARGIN=*tdistance*, and
 BOTTOM_MARGIN=*bdistance*
 options to the rule. Specify all *xdistance* parameters as positive real numbers of user units. Using an *xdistance* of 0 for any of the margin parameters is acceptable.

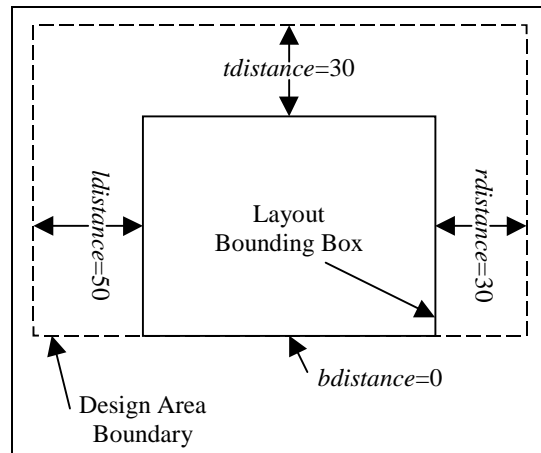


Figure 156: Using 4 margin parameters

Example: **ERR = MIN_FILL {C .25
 LEFT_MARGIN= 50
 RIGHT_MARGIN= 30
 TOP_MARGIN= 30
 BOTTOM_MARGIN=0
 }**

The rule above will check that shapes on layer C cover at least $\frac{1}{4}$ of the design area. The design area in this case is considered to start 50 user units to the left of the left design boundary, 30 user units to the right of the right design boundary, 30 user units above the top design boundary, and right at the bottom design boundary.

MIN_NOTCH

Find small notches

error_layer = **MIN_NOTCH** (*layer1*, *min_width*, [/LENGTH=*length*] [/[~]DET])

See page 106 for more information on how the DRC defines a notch.

When this rule is used, notches less than *min_width* in shapes on *layer1* will be marked with error wires on layer *error_layer*.

Example:

B=MIN_NOTCH (A, 2)

This rule will find all notches less than 2 units wide and mark them with error wires on layer B. When this rule is run on the shape in Figure 157, the top two notches will be marked with error wires. The bottom notch, which is exactly 2 units wide, will not be marked as an error.

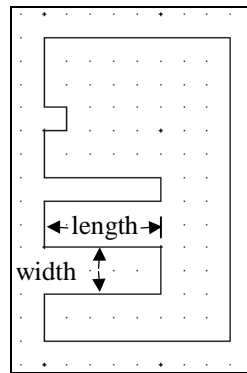


Figure 157: Shape on layer A with 3 notches.

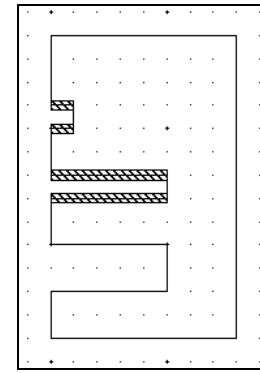


Figure 158: Notches less than 2 units wide marked with error wires.

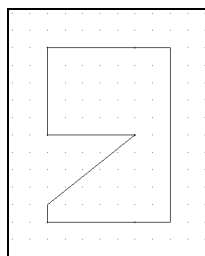


Figure 159: Angular notch will not be found.

To be recognized as a notch, opposite sides of the notch must not meet. Angular notches like the one shown in Figure 159 will **not** be found by this rule. You can use the MAX_ANGLE rule to find notches like these. However, the sides of the notch do not need to be horizontal or vertical. The portion of the notch shown in Figure 160 that is narrower than 2 units will be marked as an error.

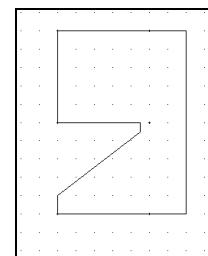


Figure 160: Error will be found.

The MIN_NOTCH rule can be a very important addition to a MIN_SPACING rule when you need to find spacing errors between shapes on the same layer. Consider Figure 161. The long wire folds back on itself and two sides are very close each other. This is a notch in a single shape rather than a spacing error between shapes.

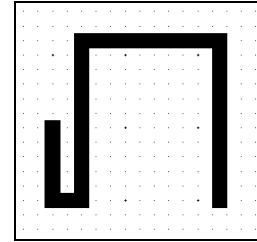


Figure 161: Notch, not MIN_SPACING error.

See an example that demonstrates this idea in NOTCHSP.RUL and WIRECEL.CEL.

A MIN_SPACING rule will **not** mark this as an error. If your design rules consider this an error, you should add a MIN_NOTCH rule to find such errors.

Remember that all touching shapes on a single layer are merged during DRC preprocessing. So even if a spacing problem like the one in Figure 161 is caused by two separate wires on the same layer, to the DRC it will be a single shape with a notch rather than a MIN_SPACING error.

The optional /LENGTH=*length* parameter is used to restrict the errors found to those at least as long as *length*. Notches less than this length will not be marked with error wires on *layer1*.

Example:

C=MIN_NOTCH (A, 2 /LENGTH=3)

When the /LENGTH keyword is added to the rule above, notches less than 3 units long will not be considered errors.

See page 50 to learn more about detailed error message in the log file.

The optional /DET keyword is used to add a detailed error message to the log file for each notch that fails the test. The coordinates of pairs of sides that fail the test will be listed. For large designs, these messages may make the log file unreasonably long.

Notches that have been discarded due to a /LENGTH restriction **will be** listed in these detailed messages.

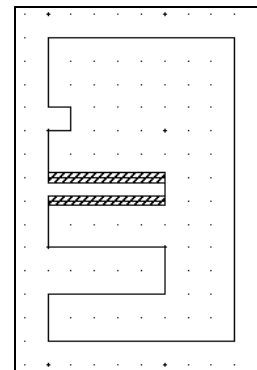


Figure 162: The notch shorter than 3 units is not marked.

DRC Rules Syntax: MIN_NOTCH

Add the /~DET option to the rule when detailed error messages have been enabled, but you want to disable them for only this rule.

MIN_SIDE*Find shapes with at least one small side*

error_layer = **MIN_SIDE** (*layer1*, *min_length*)

This rule will create error wires on all polygon sides less than *min_length*. Only the polygons on *layer1* will be tested. The error wires are created on *error_layer*.

Example: **B=MIN_SIDE (A, 2)**

All sides of polygons on layer A that are less than 2 units long will be marked with error wires on layer B. When this rule is run on the polygon in Figure 163, error wires will be created on layer B as shown in Figure 164.

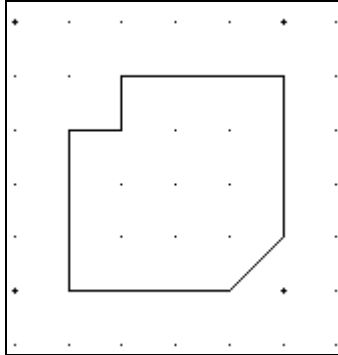


Figure 163: Polygon on layer A.

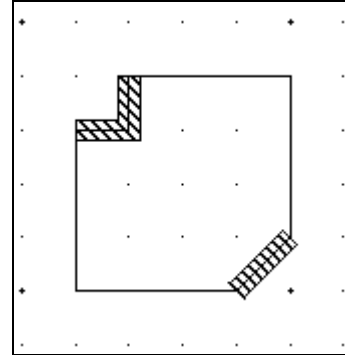


Figure 164: Error wires on layer B for all sides less than 2 units long.

MIN_SPACING

Find spacing errors

```

error_layer = MIN_SPACING ( ...
    ... from_layer [/IN] [/OUT] [/AWAY=a_angle1[/SIDES_BACK=n1]]15 [/CAP=c_angle1], ...
    ... to_layer   [/IN] [/OUT] [/AWAY=a_angle2[/SIDES_BACK=n2]]15 [/CAP=c_angle2], ...
    ... distance ...
    ... [/[~]CROSS] ...
    ... [/[~]PERP] ...
    ... [/[~]T] ...
    ... [/[~]OVER] ...
    ... [/[~]END] ...
    ... [/[~]INTER] ...
    ... [/[~]CONN] ...
    ... [/LENGTH=length] ...
    ... [/[~]DET] )

```

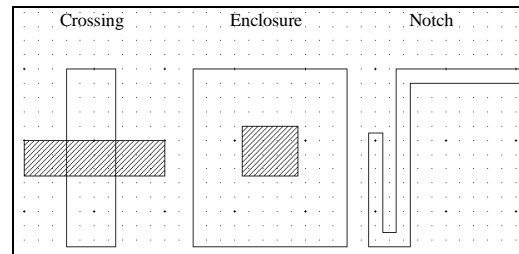


Figure 165: Configurations like these are not automatically considered MINSPACING errors.

See the table on page 26 for a list of cell and rule files included in the installation that demonstrate the MIN_SPACING rule.

This rule will find sides of shapes on the *from_layer* that are too close to sides of shapes on the *to_layer*. Set the minimum valid *distance* for the spacing check as a positive real number of user units.

This is easily the most complicated DRC rule and it is easy to write too simple a rule that will not find all of the errors you think it will. You **must** read "Spacing Verification" beginning on page 84 to learn how to write MIN_SPACING rules that catch all possible errors.

One common problem that is often overlooked by writers of DRC rule sets is that **overlapping shapes are not automatically considered errors**. Errors will be found only when a vertex of a shape on one layer is closer than *distance* to a side of a shape on the other layer. If overlaps are always errors, use the AND or TOUCHING rules to find them.

¹⁵ The AWAY option is available in beta test versions only. See page 256.

One other class of potential errors that the MIN_SPACING rule will not mark are spacing violations between parts of a single shape, or between two shapes on a single layer that have been merged during preprocessing. You will need to add a MIN_NOTCH rule to find these types of problems. See the information on page 87 and the example covered in the files NOTCHSP.RUL and WIRECEL.CEL.

Example:

ERR = MINSPACING (A, A, 2)

To classify shapes by distance rather than find errors, see the MAX_SPACING rule.

In this rule, the *from_layer* is the same as the *to_layer*. When this rule is executed on the shapes in Figure 166, error wires are created on layer ERR wherever a side of a shape on layer A is **less** than two units away from a side of another shape on the same layer. Note that no error is indicated for the bottom shape since it is exactly two units distant.

The error wires wrap around the corners of the upper two shapes. When the error extends around connected sides, the DRC will create one continuous error wire.

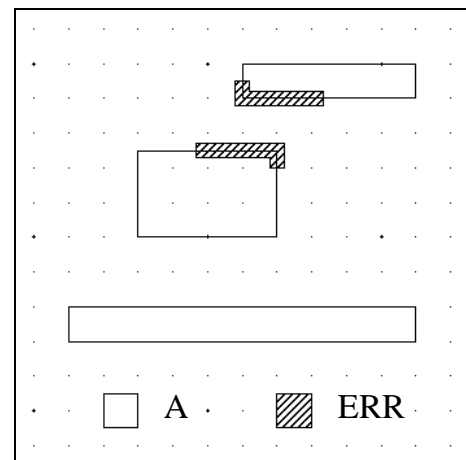


Figure 166: Error wires created for layer A shapes closer than 2 units.

Note that the underscore ('_') is optional in the MIN_SPACING keyword.

Directional Spacing Checks

The /IN and /OUT specifications are mutually exclusive for each layer. If you follow a layer name with the /IN keyword, the DRC will look only toward the inside of a shape when looking for spacing violations of the other layer. If you instead add the /OUT keyword after the layer name, the DRC will look only toward the outside of the shape for spacing violations.

When you use neither keyword, shapes on both sides of each edge of the polygon will be verified for spacing violations. We refer to this type of rule as a **simple spacing check**.

When you do use /IN or /OUT after one or both layer names, the rule is a **directional spacing check**.

Example:

ERR=MINSPPACING(A, B/IN, 1)

When this MIN_SPACING rule is run the shapes shown in Figure 167, error wires are created as shown. The DRC will look only toward the inside of the shape on layer B from each side when searching for sides of shapes on layer A that may be too close. Shapes 1 and 2 will not be found. For shape 3, only the side that is toward the inside of the side of the shape on B is indicated as an error. Each error wire for the layer A sides is paired with an error wire on the sides of the B shape.

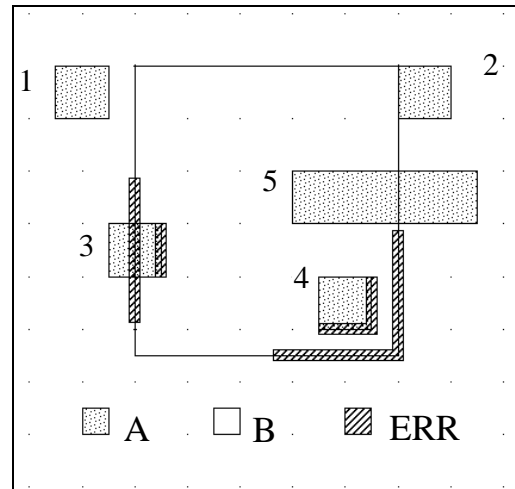


Figure 167: Error wires created from B/IN directional spacing check.

Note that shapes 3 and 4 have perpendicular sides that are also in violation, but are not marked. The default behavior for checking perpendicular or crossing sides changes when you add the /IN or /OUT keywords. We cover this subject later when we cover the orientation keywords.

Shape 5 is a special case. The horizontal sides of shape 5 are too close to the B shape, however, no vertex of the B shape is too close to these sides. Also, no vertex of shape 5 is too close to the B shape. If you consider shape 5 to be an error, you must find it with a different method. (See the examples on page 85.)

Example: **ERR=MINSPECING(A, B/OUT, 1)**

When we replace the /IN keyword in the previous example with the /OUT keyword, only sides of shapes on A that are found looking towards the outside of layer B shapes will be found. Note that the violation of shape 2 is found even though the space between the sides is exactly 0.

To get the exact coordinates for each side of a pair of sides in violation, add the /DET option to the rule. See page 269.

The violation of shape 2 is unusual for another reason. Since the error wires for each side of the violation overlap, the DRC will merge them and create only one error wire rather than a pair.

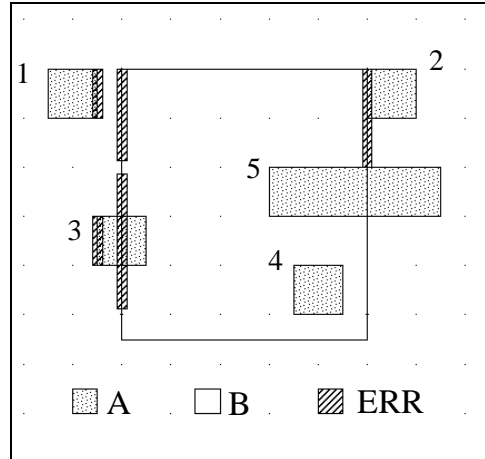


Figure 168: Error wires created from B/OUT directional spacing check.

See several examples of how to test coincident edges beginning on page 85.

You must be careful when writing a directional spacing check if you want to find shapes with coincident sides. Coincident or overlapping sides are a special case for directional spacing. Ordinarily, the MIN_SPACING rule is concerned only about side-side relationships, but when directional criteria are applied, the area of the polygons is also important. Adding the /OUT keyword to a layer specification means that area on the other layer must be present outside a shape for possible errors to be considered.

Example: **ERR=MINSPECING(A/OUT, B/IN, 2)**

When testing for possible violations on the lower shape on layer A in Figure 169, the overlapping edge is not marked. When looking to the outside of edge 3 on the lower shape, the DRC sees no material on layer B.

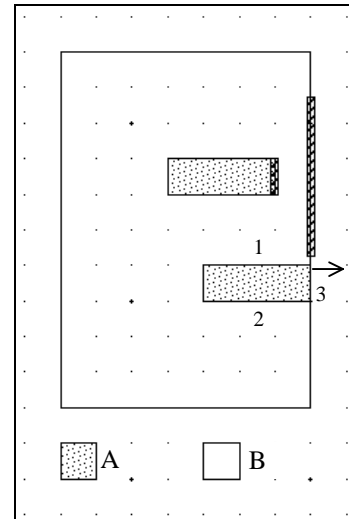


Figure 169: When A/OUT specification is used, lower shape is not marked with errors.

AWAY Option to test side-side angle – Beta test only!

New with beta version 113.65 of the DRC is the AWAY option of the MIN_SPACING rule. The AWAY option restricts errors to non-overlapping pairs of sides that are less than a certain angle apart.

The AWAY option should be added to only one layer specification. We will call this layer the *away_layer* in this discussion.

Side-side pairs that are within *distance* of each other will be **not** be marked as errors when **both** of the following conditions are met:

- 1) The side on the *away_layer* is within the specified number of sides away from the intersecting side on the other layer that is too close.
- 2) The angle between the sides is greater than the specified angle.

Let us consider the example shown in Figure 170. (This geometry is stored in the AWAY.CEL file supplied with the beta version update.) Let us assume that we need to find sides of layer B that are within 20 user units of sides of layer A shapes. However, we consider only parallel sides within this distance as true errors. We want to mark only sides 3 and 4. Perpendicular side pairs and sides at a 45° angle are permitted and should not be marked as errors.

The perpendicular sides 5 and 6 could be prevented from being marked as errors by adding the /~P option to the rule. You could prevent marking the crossing sides

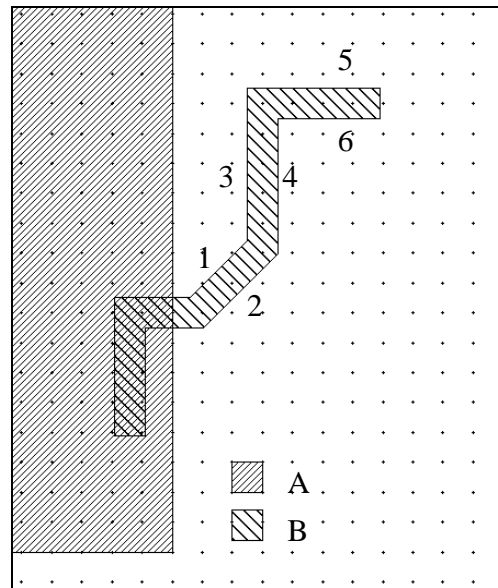


Figure 170: AWAY.CEL

with the /~CROSS option. The layer B sides within the layer A shape can be prevented from being considered as errors by adding the /OUT option to the layer A specification in the MIN_SPACING rule.

Without the AWAY option, there is no way to prevent sides 1 and 2 from being marked as errors. However, when we write the MIN_SPACING rule as follows (as shown in the AWAY.RUL file distributed with the beta version update) the DRC will mark only sides 3 and 4.

```
ERR= MIN_SPACING(A, B/AWAY=44.9, 20);
```

Since the SIDES_BACK option is not included in this example, the default of SIDES_BACK=1 will be used. This means that only sides of layer B shapes that share a vertex with a side that intersects a layer A side will have the AWAY test applied to them.

Sides that pass the SIDES_BACK test will have their angle to the side on the other layer tested. Side-side pairs will **not** be marked as errors if the angle between the side is greater than 44.9°.

Even when you want to exclude sides with angles that are exactly 45°, you should specify the AWAY angle slightly less than that number. Floating point calculations that determine the angle in the layout may result in a number slightly less than the exact angle.

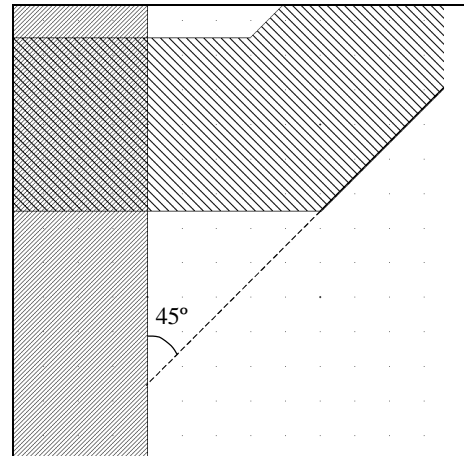


Figure 171: Angle between layer B side 2 and layer A side.

AWAY Automatic Extra Options

Whenever you use the AWAY option, the /OUT option is added to the other layer specification. This means that only sides of the *away_layer* that are found looking out from the other layer will be considered as potential errors. You cannot add the /IN option to the other layer specification when the AWAY option is used.

The /~P/~CROSS/~T options are also added automatically to any MIN_SPACING rule with an AWAY option. This means that perpendicular sides and intersecting sides are excluded as errors by default when the AWAY option is used. You can override these defaults by adding /P, /CROSS, and/or /T options to the MIN_SPACING rule.

Beta Test Warning

The new algorithms required to implement the AWAY option required changes to the MIN_SPACING algorithms. You should verify the results of all MIN_SPACING rules tested with this beta version. This includes the results of MIN_SPACING rules that do not use the AWAY option. Verify all MIN_SPACING results produced by this beta version against the results of the released version.

If the results of any MIN_SPACING rules are different between the versions, please contact IC Editors.

We do run a test suite comparing the new and old versions before we post a beta version. But just because our cases worked, that doesn't mean yours will.

End Caps

Refer to page 95 for a more complete explanation of end caps.

The `/CAP=angle` parameters are used to exclude from the spacing check all or part of the end cap of each edge. The angle parameter(s) must be between 90° and 180°. When you do not add the `/CAP` keyword to a layer, the entire end cap will be checked.

Example:

ERR=MINSPECING (A/CAP=90, B, 2.5)

When this rule is run on the shapes in Figure 172, the end caps of the shape on layer A will not be checked. Even though the shape on layer B is closer than 2.5 units, it is not in the region checked. No violations will be marked.

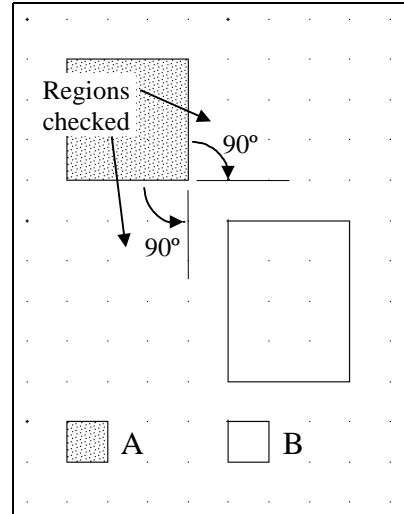


Figure 172: The end caps of the shape on layer A will not be checked.

Orientation Options

See page 97 for more details on the orientation options.

The following optional keywords are used to prevent side-side pairs in certain orientations from being considered errors. Only pairs of sides that are error candidates after the directional criteria (`/IN`, `/OUT`, and `/CAP`) are applied are considered. Then, the orientation criteria are applied to error candidates. Only pairs of sides that are in the special orientation are affected by the restrictions.

When you add a tilde ('~') in front of the option, the DRC will not consider a pair of sides to be in error when they have the indicated relationship. Options with a '~' override conflicting options.

Each of these orientation options is set for every `MIN_SPACING` rule. When you do not specify the setting in the rule, the default is used. The primary purpose of specifying orientation options is to prevent false errors from being marked. However, you may need to override the defaults in some `MINSPECING` rules to prevent the DRC default behavior from preventing real errors from being found.

	Crossing	Perpendicular	T-intersection	Overlapping	End-to-end
Simple spacing	/CROSS	/PERP	Same as crossing option	/OVER	/END
Directional spacing	/~CROSS	/~PERP		/OVER	/END

Figure 173: Default orientation options.

The orientation options in effect for each MIN_SPACING rule are always listed in the rules compiler log. If you add the LIST_RULES option to the DRC command line (see page 350), this information will also be listed in the DRC log file.

When typing a MINSPACING rule, you can use the first letter of any orientation option keyword instead of typing the entire keyword.

Remember that the **vertices** of crossing sides must be closer than *distance* for the DRC to find the violation.

/CROSS and /~CROSS

The /CROSS keyword is used to consider as errors spacing violations that involve crossing sides. This is the default behavior when the rule is a simple spacing check. The /~CROSS keyword prevents violations between crossing sides from being considered errors. /~CROSS is the default when the rule is a directional spacing check.

Let us say that you need to find all parallel wires that are too close together, but crossing wires are acceptable.

Example:

ERR=MINSPACING(A, B ,1.1 /~CROSS)

When this rule is run on the shapes shown in Figure 174, violations between sides that cross will not be marked. Only the parallel sides that are closer than 1.1 units will be marked with error wires on layer ERR.

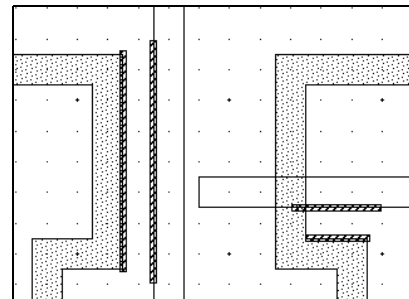


Figure 174: Only violations between sides that do not intersect are marked when /~CROSS is in effect.

Example: **ERR = MINSPACING (A, B/IN, 1.5)**

The /IN keyword after the layer B specification makes this rule a directional spacing check. The default for directional spacing checks is /~CROSS. Note that side B3 in Figure 175 crosses side A1 and violates the 1.5 spacing check. However, since the default is /~CROSS, side B3 is not marked as an error.

Side A2 is marked even though it is a crossing side. This side is marked because it violates the spacing rule with side B2, **not** the crossing side B1. Crossing sides will still be marked if they are too close to **other** sides.

When you have a side marked as an error, and the side is surrounded by other error marks, it can be difficult to determine why it violates the spacing rule. To list the specific pairs of sides that violate a spacing rule, add the /DET option (covered later) to the spacing rule.

Example: **ERR = MINSPACING (A, B/IN ,1.5 /C)**

In this example, the /CROSS option (abbreviated to /C) has been added to the MINSPACING rule above. Now, side B3 will be marked as an error since crossing sides are now considered errors.

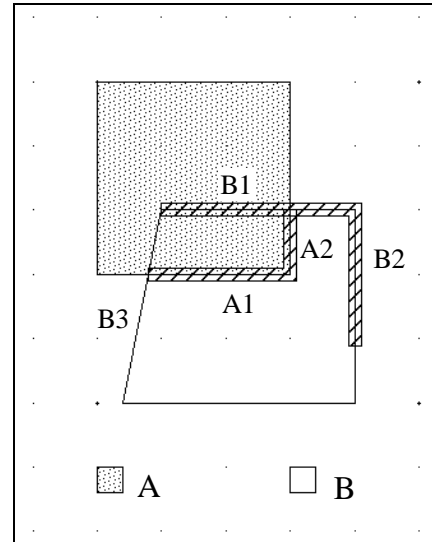


Figure 175: Crossing side B3 is not considered an error when the default /~CROSS option is used in the directional spacing check.

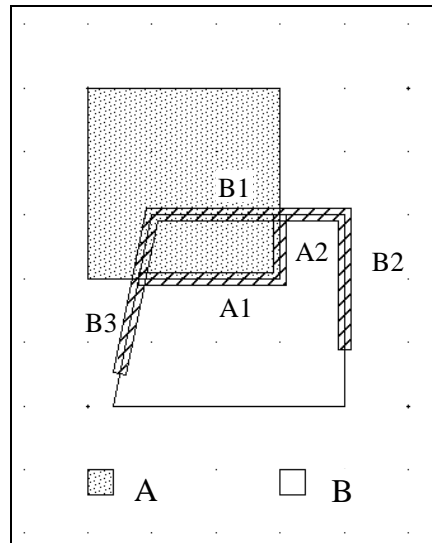


Figure 176: Crossing side B3 is marked as an error when the /C option is added to the rule.

You can always look at the rules compiler log file to how each of the orientation options is set for a particular rule. For example, the following is an excerpt from a compiler log file for the MINSPACING rule above.

```
ERR[3] = MIN_SPACING(A[1], B[2]/In, 1.5  
/+~CONN/~P/OVER/CROSS/T/END/~DET)
```

/T and /~T

Use the /T keyword when you want sides that form a T-intersection to be considered errors. The T-intersection does not need to be perpendicular. The DRC defines a T-intersection as an intersection where a single vertex of a side is on the other side, and the sides do not overlap. Use /~T when you want to prevent sides that touch with a T-intersection from being indicated as errors.

The T-intersection options are most useful when combined with other orientation options. See the example on page 267.

The default used by the DRC for T-intersections is /T if the /CROSS option is in effect. If /~CROSS is in effect, then the default option is /~T. When you override the default, the specification with the '~' overrides a conflicting option. Combining /~CROSS with /T means that crossing T-intersections will **not** be considered errors. Only side-side pairs where the end points meet but the sides do not overlap or meet end-to-end will be marked.

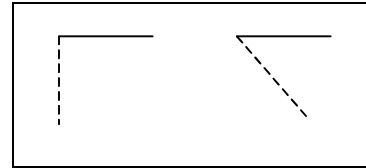


Figure 177: T-intersections without crossing sides.

/PERP and /~PERP

These options control perpendicular orientations. When you use the /PERP option, spacing violations of sides that are perpendicular will be considered errors. This is the default for simple spacing checks. When you use /~PERP, these violations are not considered errors. /~PERP is the default for directional spacing checks.

Example: **ERR = MINSPACING (A, B/IN, 1 /PERP)**

In this example, we have overridden the default /~PERP option used for directional spacing checks. The shapes are the same as those used for the example on page 254. Note that the perpendicular sides of shape 4 are now marked as well. However, the perpendicular sides of shape 3 are not marked, since they also cross the sides of the B shape. Remember that the default for directional spacing checks is /~CROSS.

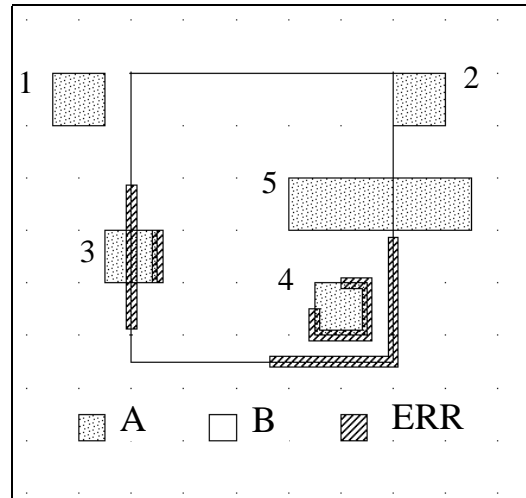


Figure 178: Error wires created from B/IN 1 /P directional spacing check.

Side-side pairs that have more than one orientation relationship will be marked as errors only if none of those relationships are disabled with a '~' in front of the option.

Note that unlike the other orientation options, the perpendicular option also regulates spacing violations of sides that do not touch.

/OVER and /~OVER

Adding /OVER to a rule has no effect since this is always the default behavior.

The /OVER option is used to consider spacing violations of overlapping sides as errors. This is the default for all MIN_SPACING rules. Use the /~OVER option to prevent overlapping sides from being considered errors.

For the next example, look at Figure 179. Let us say that shapes on layer A that are covered by layer B must be at least 1 unit away from an edge of a shape on layer B. However, layer A shapes that are coincident with an inside edge of a shape on B are valid. Shapes that have a gap larger than 0 but less than 1, are in error. Shapes on layer A are not allowed cross edges of shapes on layer B. We want to find errors like shapes 2, 4 and 5.

Example: **ERR=MINSPPACING(A, B/IN, 1)**

When the DRC executes the above rule on the shapes in Figure 179, the error wires shown in Figure 180 are generated. Shape 3 is marked with an error since it violates the 1 unit spacing rule. When you want to exclude sides that overlap as errors, you must add the /~OVER option to the rule.

Example: **ERR=MINSPPACING(A, B/IN, 1 /~OVER)**

See several examples of how to test coincident edges beginning on page 85.

When the /~OVER option is added to the rule, the DRC marks only shapes 2 and 4 as shown in Figure 181.

Note that no crossing or perpendicular sides are marked by either rule. This is because the default for directional spacing checks is /~CROSS/~PERP.

Shape 5 is not marked by either rule. This is because no vertices of shape 5 are within 1 unit of a side of the B shape. Adding /CROSS or /PERP to the rule will not solve this problem. This shape should be found with a different method. (See page 85.)

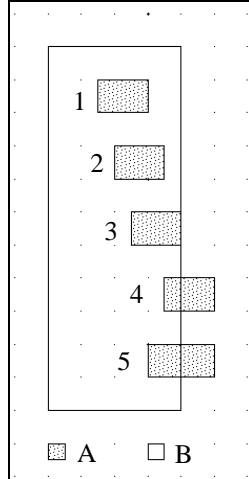


Figure 179: Only shapes 1 and 3 are valid.

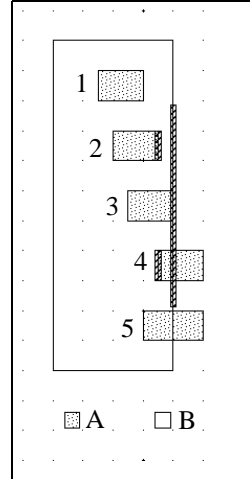


Figure 180: Shapes 2, 3, and 4 are marked with errors.

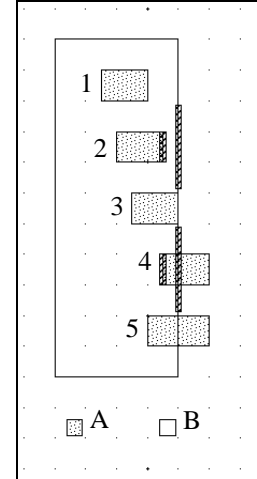


Figure 181: Only shapes 2 and 4 are marked when /~OVER is added.

/END and /~END

The /END option allows end-to-end sides to be considered errors. The sides must share a vertex and meet at 180°. /END is always the default. Add /~END to the rule if you do not consider sides that meet end-to-end as errors. Most combinations of shapes that have sides with an end-to-end relationship also have side-side pairs that have other relationships. This is why /~END is usually used in combination with other options. See the example on page 267.

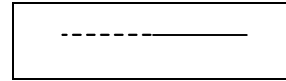


Figure 182: End-to-end intersection.

/INTER and /~INTER

These options control intersections. They are used as a quicker way to set the /CROSS and /OVER options. The /INTER (or /I) option is shorthand for the option combination /CROSS/OVER. The option /~INTER (or /~I) will set the /~CROSS and /~OVER options.

Combining Orientation Options

Let us cover a few examples of combining orientation restrictions in a MIN_SPACING rule.

Example: **ERR=MINSPPACING(A, B/IN, 1 /C/P)**

When we add both the /C and /P keywords to this MIN_SPACING rule, the horizontal sides of shape 3 in the example used previously are marked. We need to add both keywords to this directional spacing check to find perpendicular sides that cross. This is because the default for directional spacing checks is /~CROSS/~PERP. Both defaults must be overridden.

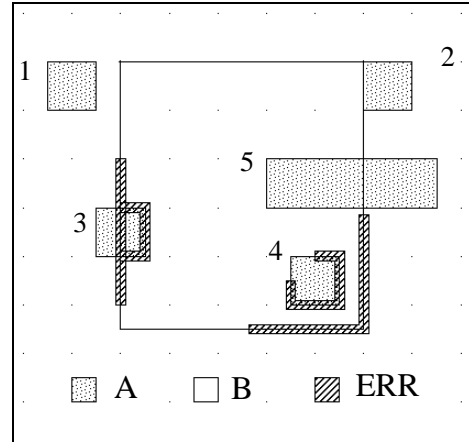


Figure 183: Error wires created from /C/P directional spacing check.

Example: **ERR=MINSPPACING(A, B, 1 /~O)**

When this rule is run on the shapes in Figure 184, portions of sides where the only relationship between them is an overlap are not considered errors. However, portions that have a T-intersection relationship are still marked as errors.

For example, side 1 on the B shape and side 2 on the A shape are marked as errors since /T is the default for simple spacing checks.

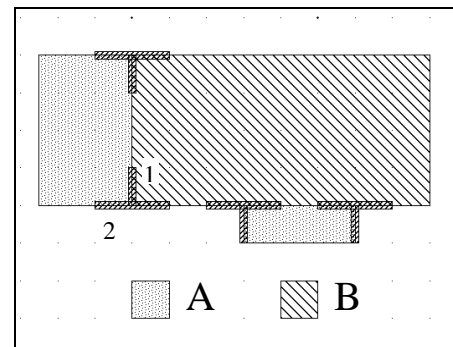


Figure 184: Errors found when /~O used.

Example: **ERR=MINSPPACING(A, B, 1 /~O/~T)**

When we add the /~T option to the same rule, this class of errors will no longer be marked. See Figure 185. However, note that some errors are still marked. These errors are indicated where sides on each layer meet end-to-end. If you want to avoid marking sides with this relationship as errors, add the /~E option (shorthand for /~END).

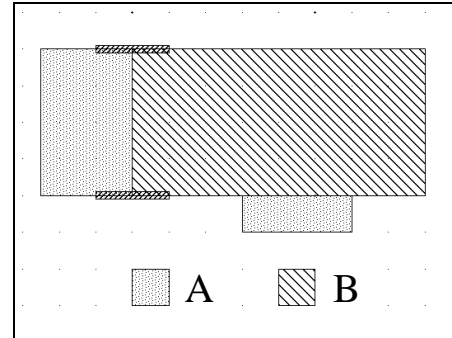


Figure 185: Errors found when /~O/~T used.

Example: **ERR=MINSPPACING(A, B, 1 /~O/~T/~E)**

When we add the /~E option to the rule used above, no errors are marked when the rule is executed on the shapes in Figure 184.

Electrical Connections

When the rules compiler or DRC lists a MIN_SPACING rule with the default connection restriction "/+~CONN", this means both connected and unconnected pairs of shapes will be checked.

Add the /CONN option to the MIN_SPACING rule if you want to restrict the spacing violations to those between electrically connected shapes. The /~CONN option will consider as errors only spacing violations between shapes that are not electrically connected. The default is always to check both connected and unconnected pairs of shapes.

For MIN_SPACING rules to accurately recognize what shapes are electrically connected, you must define how electrical connections are made. You use CONNECT and STAMP rules to define electrical connectivity. See page 110 for a complete explanation.

Example: **CONNECT A B BY C**
ERR=MINSPPACING(A, B, 2 /~CONN)

The CONNECT rule above indicates that shapes on layers A and B are electrically connected by shapes on layer C. Since the /~CONN option is used,

DRC Rules Syntax: MIN_SPACING

only spacing violations between shapes on A and B that are **not** electrically connected will be considered errors.

Example: **ERR=MINSPECING(A, B, 2 /CONN)**

See an example of setting up electrical connections for this option on page 402.

When the /CONN option is instead, it changes the rule so that only spacing violations between shapes that are on the same electrical net will be marked.

NOTE: The restrictions imposed by the /CONN or /~CONN options will be ignored when you specify the QUICK_PASS option on the DRC command line. **You should use the SLOW command line option to enable the /CONN or /~CONN options.** See page 129 for more details.

Other Options

/LENGTH=length

Add this option to the MIN_SPACING rule when you want to restrict the error wires on layer *error_layer* to those at least *length* units long.

The /LENGTH option can result in unpaired error wires. It may be difficult to determine which shape caused a spacing error when you can see only one unpaired error wire.

Example: **ERR = MINSPACING (A, B/IN, 1 /LENGTH=3)**

When the /LENGTH=3 option is added to the MIN_SPACING rule used in the example on page 254, error wires less than three units long will not be created on layer ERR. When the rule is run on the same shapes as those in Figure 167, only the error wire shown in Figure 186 will be created. The other error wire of the pair (the one on the A shape) is missing.

You will not be warned if spacing errors have been ignored due to a /LENGTH restriction.

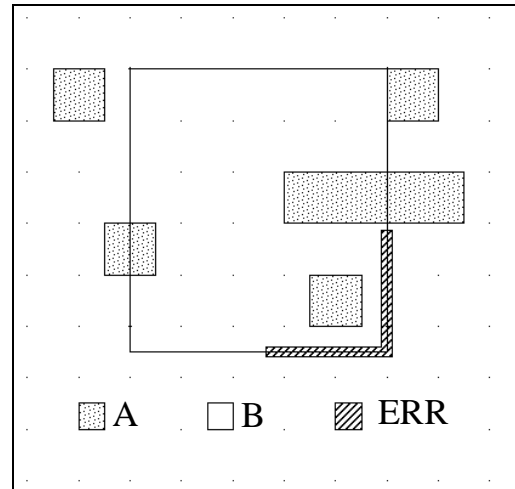


Figure 186: Error wires created from B/IN directional spacing check.

Using this option can prevent the DRC from automatically using the faster quick_spacing algorithm. This may result in longer run times. If you force this algorithm to be used by specifying the QUICK_SPACING option on the DRC command line, you may prevent the DRC from finding real errors. The log file will include a warning about this. See page 100.

See page 50 for more information on detailed logging.

/DET and ~DET

Add the /DET option to your MIN_SPACING rule to create detailed error messages in the log file for each pair of sides in violation. Detailed logging can result in **very** large log files.

Example: **ERR = MINSPACING(A, B/IN, 1.5 /DET)**

When we add the /DET option to the rule in the example on page 261, the DRC log file will contain the following text that clearly indicates each pair of sides in error. This makes it clear that side A2 in Figure 175 is in violation with side B2, not the crossing side B1.

<pre>3. RESULT1[50] = MIN_SPACING(A[1], B[2]/In, 1.5 /+~CONN/~P/OVER/~CROSS/~T/END/DET) 1: 1 (178,10)-(178,7) <-> 1 (179,8)-(179,5) 2: 1 (175,7)-(178,7) <-> 1 (176,8)-(179,8)</pre>
--

Figure 187: Example of detailed logging for a MINSPACING rule.

Add the /~DET option to a rule if detailed logging is enabled in your rule set, but you want to disable it for only that rule.

If you have a /LENGTH=*length* option in your MIN_SPACING rule, and detailed logging is enabled, the log file **will** contain details on error wires that have been discarded due to the length restriction.

MIN_WIDTH**Find shapes with small width**

error_layer = **MIN_WIDTH** (*layer1*, *min_distance*, [/LENGTH=*length*] [/[~]DET])

See page 103 for more information on how the DRC defines width.

This rule will mark as errors any sides of a polygon that are less than *min_distance* away from another side of the **same** polygon. Sides that touch are not considered errors. Only shapes on *layer1* are tested. The error wires will be created on layer *error_layer*.

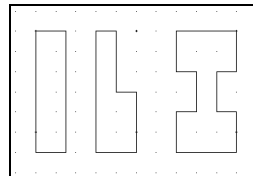


Figure 188: Shapes on layer A.

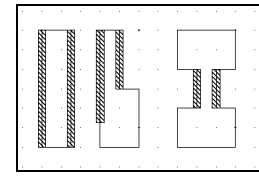


Figure 189: Sides that are closer than 2 units are marked with error wires on layer B.

Example: **B=MIN_WIDTH (A, 2)**

This rule will find sides of shapes on layer A that are closer than 2 units from other sides of the same shapes and mark them with error wires on layer B. When this rule is run on the shapes in Figure 188, each pair of sides in violation of the rule is marked with error wires on layer B as shown in Figure 189.

The optional /LENGTH=*length* parameter is used to restrict the errors found to those at least as long as *length*. Violations less than this length will not be flagged as errors.

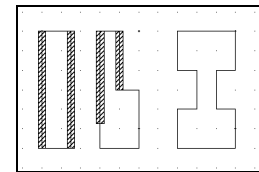


Figure 190: The width violation shorter than 3 units is not marked.

Example: **C=MIN_WIDTH (A, 2 /LENGTH=3)**

When the /LENGTH keyword is added to the rule above, violations less than 3 units long will not be considered errors.

DRC Rules Syntax: MIN_WIDTH

This rule will not locate notches. The *min_distance* is measured only across the interior of polygons. See the MIN_NOTCH rule to locate notches.

See page 50 to learn more about detailed error message in the log file.

The optional /DET keyword is used to add a detailed error message to the log file for each pair of sides that fails the test. For large designs, these messages may make the log file unreasonably long.

Add the /~DET option to the rule when detailed error messages have been enabled, but you want to disable them for only this rule.

MODIFY LAYER *Define layer used as both an input and output layer*

MODIFY LAYER *iced_layer_number drc_layer_name*

See an overview
of layer
definition rules
on page 55.

Use this rule to define layers that can be used as both input and output layers. Modify layers are useful when you are using the DRC to modify layers in a cell. Use INPUT LAYER and OUTPUT LAYER definitions instead when you are using the DRC to check for errors.

Use caution with layers defined with this rule. If you use the DRC generated command file to read shapes on modify layers into your original cell(s), you will alter existing layers. Read this entire description before using the rule.

The *iced_layer_number* parameter indicates the number of the layer in the ICED™ cell(s). The *drc_layer_name* parameter defines the name of the layer used in the rest of the DRC rule set.

Example: **MODIFY LAYER 1 A**

This rule is roughly equivalent to the following pair of rules.

INPUT LAYER 1 A
OUTPUT LAYER 1 RESULT

In either case, the shapes on layer 1 in the input data will be used as layer A in the DRC run, and shapes will be created on layer 1 in the command file the DRC generates at the end of the run. (This command file can be used for input into the ICED™ layout editor.)

The primary difference between the two definitions is that when MODIFY LAYER is used, you refer to layer 1 throughout the rule set as layer A, while the pair of INPUT LAYER and OUTPUT LAYER rules allow you to refer to layer 1 by two different names during the rule set.

DRC Rules Syntax: MODIFY LAYER

If you import the output shapes into your original cells, layer 1 may be corrupted because you now have both the old shapes on layer1 and the new ones created by the DRC run. **You must be very careful to either remove the old shapes on layer 1 before adding the new shapes to your cell, or to use the output data to create new cells.**

Since this is a hazardous operation, the DRC rules compiler will warn you when you have used the same *iced_layer_number* in both an INPUT LAYER rule and an OUTPUT layer rule. However, when you use MODIFY LAYER, the DRC rules compiler will assume that you know what you are doing and will **not** issue a warning.

Let us say that you have library of cells and you need to shrink the size of all metal wires in these cells. You want to create new copies of all cells in the library with this change made. You can perform this function easily with the DRC.

Example:

```
ALL_SAFE  
MODIFY LAYER 1 M1; 2 M2; 3 POLY; 4 DIFF;  
MODIFY LAYER 5 CONT; 6 VIA; 7 WELL;
```

```
M1 = SHRINK(M1, .2)  
M2 = SHRINK(M2, .2)
```

The command file created when you run this set of rules on a cell will include the data on all seven layers listed in the MODIFY LAYER rules, even though five of those layers are unchanged by the rule set. You can create a new, empty cell with the ICED™ layout editor and run the command file to create the shapes for the new cell.

There are a few things you should be aware of when you use a process like this. All wires will be converted to polygons in the output data. All text and line components will be ignored.

Refer to page 134 for details on the relationship between hierarchical output and dangerous processing options.

If you add all original cells to a main cell and create the data for the DRC from this main cell, you can use the `HIERARCHICAL` command line option to process all cells with one DRC run. In this case, you will want to change `ALL_SAFE` in the rule set above to `ALL_DANGER`.

The syntax for defining multiple layers is the same as that used for the `INPUT LAYER` rule. See page 221 for examples.

NO_CHECK_INPUT *Prevent some bad polygons from being marked*

NO_CHECK_INPUT

See page 74 to learn more about bad polygons.

By default, the DRC will search for bad polygons on all layers defined with INPUT LAYER or MODIFY LAYER rules. This is true even when some of those layers are not used in any processing rules. If you prefer to have the DRC ignore bad polygons on layers that are not actually used in the rule set, add this rule anywhere in your rule set.

Example:

NOCHECKINPUT

Note that the underscores are optional when typing this rule. This is true of all rules.

When this rule is not present in your rule set, bad polygons on input layers or modify layers that are not used by any rules in your rule set will be still be copied to an error layer and reported in the log file.

Even when you do add this rule to your rule set, bad polygons on input layers or modify layers that are used by other rules will be copied to an error layer and reported in the log file.

NO_HIER_WARNING

Prevent warning during hierarchical output

NO_HIER_WARNING

See page 146 to see an overview on hierarchical output.

See page 354 for details on the HIERARCHICAL command line option

This rule is useful only when you are creating hierarchical output through the use of the HIERARCHICAL command line option.

Whenever you are creating hierarchical output, and the ALL_DANGER rule is **not** used, safe processing may force some of the shapes to be created higher up in the hierarchy than you would expect. In this case, the DRC creates the following warning in the log file and posts it to the console. You must reply to the warning prompt for the run to proceed.

```
*****WARNING*** You specified hierarchical output.  
Because the rules file does not allow at least  
some operations to be done dangerously, some output  
may not be hierarchical. This can be avoided with  
ALL_DANGER in the rules file. If this results in  
wrong answers, there will be a message on your log  
file.
```

You can avoid this warning message by placing NO_HIER_WARNING in your rules file or on the command line.

To avoid the warning prompt, you can add the NO_HIER_WARNING rule to the rule set. When this is the case, no warning message will be created.

NO_PANELS

Execute DRC on entire design at once

NO_PANELS

To understand
how panels are
used, you
should read
Panel
Processing on
page 118.

If your layouts are small and simple you may want to execute the DRC on the entire design at once instead of dividing it into panels with the default DRC panel settings or the PANELX and PANELY rules. Add the NO_PANELS rule to specify a single panel that covers the entire design area.

Do not use this option on larger designs, e.g. entire chips.

NO_RUL

Prevent warning when source rules file is missing

NO_RUL

This rule in the rule set has the same effect as adding the NO_RUL option to the DRC command line.

The DRC rules compiler stores the location and time/date stamp of the source rules file in the compiled rules file. When you run the DRC, one of the first tasks performed by the program is to check the time/date stamp stored in the compiled rules file against the source rules file. The DRC will post a warning and wait for you to respond if the source rules file is different from the one used to create the compiled file, or if the source rules file can't be found. This is to avoid a wasted run when you modify the source rules file, but forget to compile it before running the DRC.

If you prefer to avoid the warning prompt when the source rules file will not be available, add this rule to your rule set. This can be especially useful when you intend to distribute a compiled rule set to others. Since the source rules set will be missing when those users use your compiled rules, they will receive the warning prompt unless you add this rule to the rule set. (The warning prompt can also be suppressed with the NO_RUL option on the DRC command line.)

This rule will not prevent the DRC from issuing the warning prompt when the source rules file is found by the DRC and it has a different time/date stamp than the compiled rules file.

NO_WARN_ACUTE

Prevent marking acute angles

NO_WARN_ACUTE

Use the
WARN_ACUTE
rule to change the
special layer
number. It is
layer 99 by
default.

Acute angles can cause problems to mask-processing software. So the DRC will by default alert you to all acute angles on output layers. All acute angles on output layers will be marked with wire shapes on a special output layer and reported in the log file. (This is a departure from earlier versions of the DRC.)

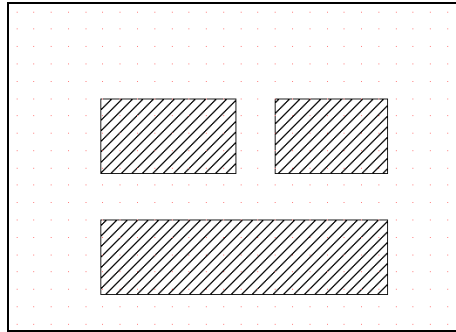
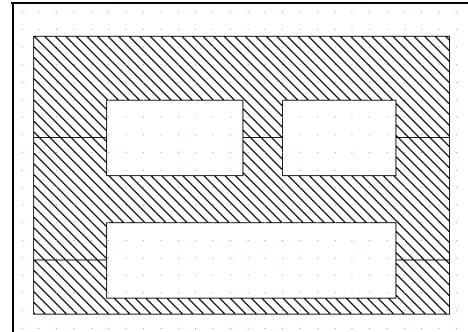
Acute angles on output layers receive this special handling due to the fact that the DRC can create shapes with acute angles in special cases. If you use the DRC to generate mask layers, you should fix these shapes by hand in the layout editor before sending the data to your foundry.

However, if acute angles are not a problem for your design, or if you are generating output layers that will not be used as mask layers, you can add this rule to your rule set to prevent both the reporting of these acute angles and the generation of the wire shapes on the special layer to mark all acute angles.

Alternately, if you want to only suppress the generation of the wire shapes, while still reporting the acute angles in the log file, use the WARN_ACUTE=0 rule instead of this rule.

NOT**Copy inverse of layer***result_layer* = **NOT** *layer1*

This rule is used to create the inverse of a layer. It is simply a form of the assignment rule (already covered on page 187) with the optional NOT included.

*Example:***NWELL = NOT PWELL****Figure 191: Layer PWELL****Figure 192: Layer NWELL = NOT PWELL**

The bounding box is the smallest rectangle, square with the axes, which encloses the design.

The rule above will create the inverse of the PWELL layer. The outer boundary of the inverse layer is slightly larger than the bounding box of your design. When the NWELL layer is used by other rules in the DRC, it will remain one large polygon with holes in it. If the NWELL layer is an output layer, before the DRC can output the layer as ICED™ components the shape must be divided into several polygons. Polygons with holes not connected to the outer boundary are not valid components in ICED™. The somewhat arbitrary cut lines (where the NWELL shape is cut to create valid polygon shapes) will have no effect on processing in the DRC.

The CUT_RESOLUTION rule is used to define the grid for cut lines when a shape with holes is cut into valid polygon shapes.

OFF_GRID

Find vertices that are not on resolution grid

error_layer = **OFF_GRID** (*layer1*, *grid_resolution*)

Refer to page 79
for an overview
of grid
resolution
issues.

Use this rule to find polygons on *layer1* containing vertices that are not on the indicated grid. Polygons with at least one vertex with a coordinate that cannot be expressed as a multiple of *grid_resolution* will be copied to *error_layer* and counted as errors in the log file.

The *grid_resolution* is expressed as a positive real number of user units.

Example:

B=OFF_GRID (A, 1)

This rule will copy to layer B polygons on layer A that have at least one vertex with a non-integer coordinate.

Example:

C=OFF_GRID (A, .1)

Resolve off-grid
problems with
the SNAP and
SNAP45 rules.

Polygons on layer A with a vertex not on a grid with .1 spacing will be copied to layer C and included in the error count.

Note On Touching Shapes

Remember that the DRC merges all touching shapes before verifying the geometry. If two shapes on *layer1* share an edge on an off-grid coordinate, but the merged shape has no edges with off-grid coordinates, the shapes **will not** be marked as errors. See the overview on page 79 for details.

Preventing Off-Grid Coordinates

See the CUT_RESOLUTION rule to prevent off grid coordinates from being created by the DRC on generated layers at panel boundaries.

OR**Boolean OR of two layers**

result_layer = [NOT] *layer1* **OR** [NOT] *layer2*

This rule will create the union of all shapes on layers *layer1* and *layer2*.

Example: **C = A OR B**

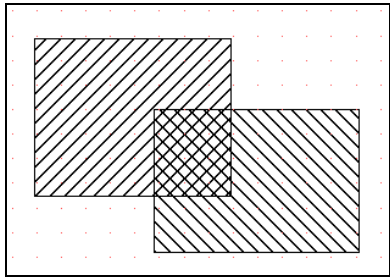


Figure 193: Polygons on layers A and B

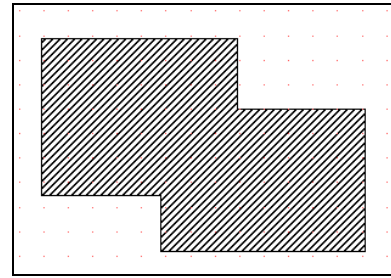


Figure 194: C = A OR B

The optional NOT keywords will perform the operation with the inverse of the layer instead of the original layer.

OUTPUT LAYER

Define layer for output

OUTPUT [ERROR] [WIRE] [POLYGON] **LAYER** *iced_layer_number drc_layer_name*

Refer to page 55 to learn more about how layers are used by the DRC.

Output layers will be included in the command file generated by the DRC. This file can be used to create shapes in ICED™ cells. Only layers defined with OUTPUT LAYER rules (or MODIFY LAYER rules) can be imported into the ICED™ layout editor. Be sure that all layers used to locate errors are defined with this rule. Use SCRATCH LAYER rules to define all other layers the DRC will create or modify.

Refer to page 70 if you using output layers to generate mask layers.

The only required parameters for the OUTPUT LAYER rule are the *iced_layer_number* and the *drc_layer_name*. The *iced_layer_number* will be the number of the layer created in the ICED™ cell when you execute the command file created by the DRC.

To see how to import these layers into the ICED™ layout editor, see page 365.

The *drc_layer_name* is the name of the layer used in the other DRC rules. The name will not be used in the ICED™ cell. Only the layer number is preserved as you import the shapes into ICED™. The data is created after all rules are executed at the conclusion of the DRC run.

Example:

OUTPUT LAYER 101 GATE

You can specify layer numbers at run time with the LAYERS option on the DRC command line. See page 346.

This example defines the layer GATE. Since GATE is an output layer, at the end of the DRC run all shapes on that layer will be included in a command file which can be used to create the shapes in an ICED™ layout editor session. The layer number in the ICED™ cell will be 101. (The layer in the ICED™ cell will **not** automatically have the name GATE. Whatever name was assigned in the cell to this layer number, if any, will remain the name of the layer.)

You can use the same *iced_layer_number* for both an input layer and an output layer, but you will receive a warning from the compiler. To avoid the warning, use MODIFY LAYER instead. (See page 273.)

You can output more than one *drc_layer_name* to one *iced_layer_number*. In this case, shapes from several DRC layers will all be created on one ICED™ layer.

Example: **OUTPUT LAYER 10 POLY**
 OUTPUT LAYER 10 RESISTOR_POLY

To assign a name to a layer in the ICED™ layout editor, use the LAYER command.

This pair of rules defines two output layers with the same *iced_layer_number*. All shapes on both DRC layers POLY and RESISTOR_POLY at the conclusion of the DRC run will result in shapes on layer 10 in the command file. The layers are processed separately during the DRC run.

You can use semicolons and curly braces to allow more than one layer definition in one OUTPUT LAYER rule. The syntax is the same as that used in the INPUT LAYER rule. See page 221 for more details.

Example: **OUTPUT LAYER {**
 11 DIFF ! diffusion layer
 12 DEV ! device layer
 }

Defining an Error Layer

Set the width of all error wires with the WIRE_WIDTH rule.

The optional ERROR keyword will cause the layer to be treated as an error layer. If shapes on the indicated layer exist at the end of the DRC run, they will be included in the error count. The number of shapes on each error layer is reported in the "Error Layer Outputs" section of the log file.

Any rule that uses the term *error_layer* on the left side of the '=' in the syntax statement **automatically** classifies the layer as an error layer. Refer to page 62 to see which rules automatically classify their result layers as error layers. You do not need to add the ERROR keyword to the OUTPUT LAYER rule for the layers created by any of these rules.

DRC Rules Syntax: OUTPUT LAYER

However, if you generate a layer you consider an error layer, but it is created by a rule that does not classify the result layer as an error layer (e.g. any of the Boolean rules: AND, OR, etc.), you should add the ERROR keyword to the OUTPUT LAYER rule so that shapes on this layer are counted as errors by the DRC.

Example: **OUTPUT ERROR LAYER 11 RESULT**
 RESULT = A AND B

All polygons on output layers will be tested for acute angles. See the information in the WARN-
_ACUTE rule.

This pair of rules will cause the DRC to create on layer RESULT the intersection of layers A and B. Since the ERROR keyword is present in the OUTPUT LAYER rule, all shapes on this layer will be included in the error count. If the ERROR keyword is not included, shapes on layer RESULT would not be counted as errors.

The WIRE and POLYGON Keywords

The MASK keyword is an obsolete, but still supported, synonym for the POLYGON keyword. Similarly, OUTLINE is a synonym for WIRE.

The WIRE and POLYGON keywords are mutually exclusive. The WIRE keyword will force the creation of wires instead of polygons. The wires will form the **outline** of polygons on the layer. The conversion takes place only when the shapes are output at the end of the DRC run.

The WIRE option **will not** transform polygons that were originally wires on an input layer back into ordinary wires on an output layer. Once the input pre-processing has transformed wires in the input data into polygons, there is no way to transform them back into ordinary wire components for output data.

The POLYGON keyword indicates that the layer should contain only polygons. If you attempt to use a layer defined with a OUTPUT POLYGON LAYER rule on the left of the '=' in any rule which creates error wires, you will get an error message from the rules compiler. Since rules that do not create error wires create polygons by default, this keyword is redundant unless you want the compiler to warn you if you are creating error wires on a layer that you consider to be a mask layer.

Defining Temporary Scratch Layers with Layer 0

The *iced_layer_number* 0 is treated differently than other output layers. Commands that create shapes on layer 0 will **not** be included in the output command file. Instead, the layer is treated as a scratch layer. This feature makes it much easier to debug rule sets.

See an example that uses layer 0 processing on page 152.

Let us say that you have an intermediate layer you need to look at occasionally to diagnose problems with your rule set. This layer is really a scratch layer and is not usually output. However you do want to include it in the output file occasionally. You should define this layer as an output layer with layer number 0. When you **do** want to see this layer in the output, simply edit the layer number to a number other than 0 and the layer will be included in the output. This is much easier than editing the rules file to move the layer back and forth from an OUTPUT LAYER statement to a SCRATCH LAYER statement.

Shapes with holes will be cut into multiple shapes on output. See page 281 for an example. The resolution grid for the cut lines is set by the CUT_RESOLUTION rule.

You can have several output layers assigned to layer number 0 (or any other layer number) and they will still be handled as separate layers during DRC processing.

OVERLAPPING

Find shapes with common area

$result_layer = layer1 [NOT]^{16} \text{ OVERLAPPING } [n1 [:n2]] layer2 [NOT=result_layer2]^{16}$

This rule is used to classify polygons on *layer1* based on whether or not they overlap polygons on *layer2*. Polygons touching only at a point, or sharing only an edge, are **not** considered to be overlapping. All shapes that overlap also touch. (See the TOUCHING rule on page 311.)

Example: **C = A OVERLAPPING B**

In this example, layer C will contain all polygons on A which overlap at least one polygon on layer B

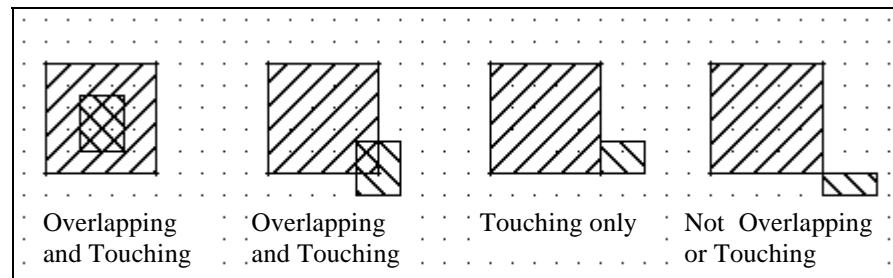


Figure 195

Only one optional NOT keyword can be used in the OVERLAPPING rule. (The NOT keywords and the *n1* and *n2* parameters work in exactly the same manner as they do in the TOUCHING rule. See page 311 for more details and more examples.)

¹⁶ Only one optional NOT keyword is allowed in a single rule.

Example: **C = A OVERLAPPING 2 B NOT=D**

This example will collect on layer C all layer A shapes that overlap exactly 2 layer B shapes. All other shapes on layer A will be copied to layer D.

PANEL_VERTICALS

Control number of vertices per panel

PANEL_VERTICALS [=] *panel_spec*

To understand how panels are used, you should read Panel Processing on page 118.

The DRC divides large layout databases into panels. This allows the DRC to process entire chips with the memory available on personal computers. The DRC attempts to calculate optimal panel size based on design size, density, and available memory. (Previous versions always defaulted to processing the data as a single panel unless the PANELX and/or PANELY rules were used.) Most designs may complete with acceptable run times with this default behavior.

If the default panel sizes do not seem to be optimal given your design and memory constraints, one option is to “tweak” the automatic panel calculations with the PANEL_VERTICALS rule.

DRC memory is divided between main memory and data storage memory. See details on page 161.

The PANEL_VERTICALS rule controls panel size by restricting the number of vertices in a single panel, rather than by specifying exact dimension as in the PANELX and PANELY rules. This makes PANEL_VERTICALS more flexible than the PANELX and PANELY rules for use in different designs and systems.

panel_spec is specified as the maximum number of relevant input layer vertices per panel per Megabyte of main memory available to the DRC, or:

$$\frac{\text{Max \# Vertices in a Panel}}{\text{Megabytes_Main_Memory}} = \text{panel_spec}$$

The default panel size is provided in your log file.

By default, *panel_spec* is set to 5000. This provides a roughly optimum number of vertices in a panel for most designs.

If you have 50 Megabytes of main memory available to the DRC, the default value for *panel_spec* results in the following equation:

$$\frac{\text{\# Vertices}}{\text{\# Panels} * 50} = 5000$$

or

$$\text{\# Vertices/Panel} = 250,000$$

If the total number of vertices in your design was 25 million, then the design would be divided into approximately 100 equal size panels.

The PANELX and PANELY rules instead set an explicit maximum panel size.

Since there is a trade off between extra processing required for panel processing and time saved due the smaller amount of data stored in flattened form at any given time, time may be saved by increasing the default panel size or by decreasing it.

- If a run with the default number of panels completes successfully, you can see if a different number of panels leads to faster run times by specifying different PANEL_VERTICALS values. The DRC log file lists the amount of time spent by each phase of the processing near the bottom of the file. If the log file indicates that the DRC is spending significant time swapping data to disk, try adding a PANEL_VERTICALS rule in your rule set with a number smaller than 5000. If the log file indicates that little or no time is spent swapping data to disk, try increasing the panel size by with a PANEL_VERTICALS rule using a value larger than 5000.
- **On the other hand if the DRC crashes with a message that indicates a memory or panel size problem, or if disk swaps are slowing your run, try a number smaller than 5000 in the PANEL_VERTICALS rule.**

You can significantly decrease the amount of time the DRC takes to complete a run by optimizing panel processing. Try various values for PANEL_VERTICALS until you come up with an optimal value for your computer and design. Set *panel_spec* to a positive real number.

DRC Rules Syntax: PANEL_VERTICES

Example: **PANEL_VERTICES = 3000**

You can try this value for PANEL_VERTICES if the DRC was unable to complete with the default and you have a very limited amount of memory on your system. If you are running a Multitasking operating system such as Microsoft Windows and you have only 32 Megabytes on your system, the DRC may have as little as 10 Megabytes of main memory available. In this case:

$$\begin{array}{rcl} & \# \text{ Vertices} & \\ \hline & \# \text{ Panels} * 10 & = 3000 \\ \text{or} & & \\ & \# \text{ Vertices/Panel} & = 30,000 \end{array}$$

PANELX and PANELY

Define maximum panel size

PANELX [=] *panel_x_dimension*

and

PANELY [=] *panel_y_dimension*

To understand how panels are used, you should read Panel Processing on page 118.

As of version 3.14, the DRC attempts to calculate optimal panel size based on design size, density, and available memory. (Previous versions always defaulted to processing the data as a single panel unless the PANELX and/or PANELY rules were used.) This automates the panel size selection process, and some designs will complete with acceptable run times with this default behavior.

The default panel size is reported in the log file. This default panel size may not be optimal for your design. You may want to optimize panel size to get faster run times by using the PANELX and PANELY rules. These rules explicitly set the maximum panel size the DRC will use. You may see improved run times with either a smaller or larger panel size than the default.

See the NO_PANELS rule to specify a single panel that covers the entire design area.

(The PANEL_VERTICES rule sets panel size according to the number of vertices and memory available rather than an explicit size. This rule is more versatile when your rule set deals with various design sizes or densities.)

The DRC reports the amount of time spent disk swapping near the end of the log file. If the DRC is spending a majority of the processing time in disk swapping, you should try reducing run time by using the PANELX and PANELY rules to force the DRC to process your design in smaller portions.

Both panel dimensions should be positive real numbers in user units. Since the DRC will divide your design into roughly equal panels, the actual size of your panels will probably be somewhat smaller than the values you set with these rules.

DRC Rules Syntax: PANELX and PANELY

Example: **PANELX = 300**
 PANELY = 300

Let us say that your design is 720 user units in the x-direction and 580 user units in the y-direction. When the above rules are used to set the panel size, the design will be divided into six 240 by 290 panels.

RULE_SET

Define sets of rules to control execution

RULE_SET *set_name_1* [, *set_name_2* [..., *set_name_10*]]

Refer to the DO
command line
parameter on
page 347.

Use this rule to define sets of rules that can be run selectively when the DRC is run. You specify which rule subsets are run using the DO parameter on the DRC command line at run time.

Example:

```
INPUT LAYER 1 M1; 2 M2; 3 DIFF; 4 POLY
OUTPUT LAYER {
    11 M1_ERROR
    12 M2_ERROR
    13 SMALL_GATE
    14 SMALL_GATE_SIDE
}
SCRATCH LAYER GATE

RULE_SET  DEVICE_RULES WIRE_SPACING_RULES

GATE = DIFF AND POLY

DEVICE_RULES ON                                ! Start of rule set
    SMALL_GATE = MIN_AREA (GATE, 4 /BORDER=4)
    SMALL_GATE_SIDE = MIN_SIDE (GATE, 1.5)
DEVICE_RULES OFF                                ! End of rule set

WIRE_SPACING_RULES ON                           ! Start of rule set
    M1_ERROR = MIN_SPACING (M1, M1, 2)
    M2_ERROR = MIN_SPACING (M2, M2, 2.5)
WIRE_SPACING_RULES OFF                           ! End of rule set
```

DRC Rules Syntax: RULE_SET

The DO command line option can also specify rules by number even when you have not defined any rule sets.

See page 151 for more details on how the DRC optimizes a rule set.

The set of rules above defines two rule subsets: **DEVICE_RULES** and **WIRE_SPACING_RULES**. You can direct the DRC to execute only the **WIRE_SPACING_RULES** subset by adding the following option to the DRC command line option:

DO=(WIRE_SPACING_RULES)

In this case, the rules that create the **SMALL_GATE** and **SMALL_GATE_SIDE** layers will not be executed. Since the **GATE** layer is no longer used in the remaining rules, the DRC will automatically skip executing the **AND** rule which generates it.

Since the DRC will determine which layer processing rules are required to execute the rules in a rule set, it is best to include only the final result rules (e.g. verification rules or output layer generation rules) in a rule set. All layer processing rules can be created earlier in the rules file, outside of any named rule set. Then create the final rules in named rule sets to be able to selectively execute them. Only the required layer processing rules for the selected rule sets will be executed and the rest will be ignored.

You can turn a rule set on and off more than once in a rules file. You can also define more than one rule subset in a single file, up to 10 rule subsets. The rule subsets may overlap, in other words a specific rule may be in more than 1 subset.

Example:

RULE_SET DEVICE_RULES FET_RULES

GATE = DIFF AND POLY
RES_POLY = POLY AND RES_MASK

DEVICE_RULES ON **! Start of rule set**
FET_RULES ON **! Start of rule set**
 SMALL_GATE = MIN_AREA (GATE, 4 /BORDER=4)
 SMALL_GATE_SIDE = MIN_SIDE (GATE, 1.5)
FET_RULES OFF **! End of rule set**
 RES_ERROR = MIN_WIDTH(RES_POLY,1.5)
DEVICE_RULES OFF **! End of rule set**

The rules that create the **SMALL_GATE** and **SMALL_GATE_SIDE** layers are contained in both rule sets.

SAFE_CELL

Flatten only certain cells for dangerous operations

SAFE_CELL *cell_name* [*cell_name_2* [...*cell_name_n*]]

You should refer to page 136 to learn about dangerous operations.

This rule specifies certain cells that the DRC should flatten before performing dangerous operations. When this rule is used, all cells not identified as safe cells will be handled in a dangerous manner (i.e. they will not be flattened).

Example:

SAFE_CELL SUBCELL

This rule will force the DRC to flatten the cell SUBCELL for dangerous operations. All other cells will not be flattened for dangerous operations.

You can supply more than one SAFE_CELL rule. You can also specify more than one cell in the SAFE_CELL rule. Simply list all required cell names on the same line. If you prefer, you can use curly braces to allow more than one line of cell specifications in a single rule. The syntax for this is the same as that used for the DANGER_CELL rule. See page 207 for examples.

The *cell_name* parameters can contain wildcard characters ('*'). When an asterisk is present, the DRC will handle as a safe cell any cell with a name that matches the given string with one or more characters replacing the asterisk. A vertical bar, '|' can be used as well to indicate a list of valid cell names. More than one '|' delimiter can be used.

Example:

SAFE_CELL AND*|*INV|*12K*

When this rule is used, all cells that begin with the string "AND" and those that end in the string "INV" will be handled safely. So will all cells that contain the string "12K" anywhere in the cell name. All other cells will be handled dangerously.

DRC Rules Syntax: SAFE_CELL

This rule is incompatible with the rules ALL_DANGER, ALL_SAFE, and DANGER_CELL. When SAFE_CELL is used in combination with SAFE_LAYER or DANGER_LAYER rules, the SAFE_LAYER or DANGER_LAYER rules take precedence. See page 142 for an example.

SAFE_LAYER

Force cell flattening for critical layers

SAFE_LAYER *layer1* [*layer2* [...*layern*]]

You should refer to page 136 to learn about dangerous operations and hierarchical processing.

Use this rule to specify layers that should be handled safely by the DRC for dangerous operations regardless of the default specification for all cells defined by the ALL_DANGER, DANGER_CELL, or SAFE_CELL rules.

Specify the names of layers that should be generated safely. You cannot specify input layers in this rule. Only the layer(s) specified in this rule will be processed safely. Other layers in cells that contain the indicated layers will not be affected.

You may want to use this rule rather than ALL_SAFE or SAFE_CELL when you have only a small area of a large cell you need to be handled safely. You can add a small shape on a dummy layer that isolates the problem shapes on a new layer that you specify in a SAFE_LAYER rule. See the example on page 142.

You can supply more than one SAFE_LAYER rule. You can also specify more than one layer in a single SAFE_LAYER rule. Simply list all required layer names on the same line. If you prefer, you can use curly braces to allow more than one line of layer specifications in a single rule. The syntax is the same as that used in the DANGER_LAYER rule. See page 209 for examples.

SCRATCH LAYER

Define temporary layer

SCRATCH LAYER *drc_layer_name*

All layers used in the rules file must be defined before they are used in a rule. This rule is used to define intermediate layers that are neither input layers or output layers. If a layer used in the rules set is not defined with the INPUT LAYER, MODIFY LAYER, or OUTPUT LAYER rules, you must define it as a scratch layer using this rule.

The use of semicolons and curly braces to allow more than one layer definition in one statement is the same as their use in the INPUT LAYER rule.

Examples: **SCRATCH LAYER SRC_DRN; GATE; POLY_WIRE;**

```
SCRATCH LAYER {  
    SRC_DRN;  
    GATE;  
    POLY_WIRE;  
}
```

```
SCRATCH LAYER {  
    SRC_DRN  
    GATE  
    POLY_WIRE  
}
```

All three of these examples are exactly equivalent. The semicolons are not required when one layer is defined on each line.

If you use a layer defined with the SCRATCH LAYER rule as the *error_layer* for any of the rules that automatically generate polygon shapes on an error layer, the shapes created will not count as errors and will not be included in the output. (This includes the MIN_AREA, OFF_GRID, and STAMP rules. See the entire list on page 62. If you do not use the layer in other succeeding rules, you will receive a warning from the rules compiler.)

For, example, let us say that shapes on layer A with a small area are not always errors. You want to classify shapes on layer A by area using a MIN_AREA rule, but you do not want to count all shapes created by the rule as errors. Since the SMALL_A layer is defined as a scratch layer rather than as an output layer, shapes on SMALL_A are not counted as errors. However, shapes on SMALL_A_NO_B will be counted as errors since SMALL_A_NO_B is an output error layer.

Example:

INPUT LAYER	1 A; 2 B;
SCRATCH LAYER	SMALL_A;
OUTPUT ERROR LAYER 90	SMALL_A_NO_B

SMALL_A = MIN_AREA (A, 6 /BORDER = 6)
SMALL_A_NO_B = SMALL_A AND NOT B

You can use this method of suppressing errors for a rule that usually creates error shapes only for rules that create polygons, not error wires.

See an example
that uses layer 0
processing on
page 152.

Another way to specify scratch layers is to use an OUTPUT LAYER rule with a layer number of 0. No shapes on layer 0 will be included in the output. This method is often more convenient than using SCRATCH LAYER rules since you can easily change any layer from a scratch layer to an output layer by editing the layer number to a non-zero number. This requires less editing than changing a layer definition from a SCRATCH LAYER statement to an OUTPUT LAYER statement. See page 287. You may often need to look at scratch layers when debugging a rule set.

SHRINK

Shrink shapes uniformly
$$result_layer = \text{SHRINK} (layer1, offset_val)$$

Use the SHRINK rule to store on *result_layer* polygons on *layer1* which have been shrunk by *offset_val*. All sides of the polygons will be shifted inwards in a parallel manner by *offset_val*. *offset_val* must be a positive real number of user units.

Example: **B = SHRINK (A, 1.2)**

Note that the parentheses and comma are required in the SHRINK rule.

Polygons can change shape significantly when being shrunk. Thin sections that become a width of zero or less will simply disappear. Small polygons with either dimension less than twice *offset_val* will disappear entirely.

See the BLOAT rule.

The DRC processes a SHRINK operation as a BLOAT of the inverse of a layer. When you shrink a shape with an acute angle notch, you are really bloating a shape with an acute angle. The bloat of an acute angle can result in significant distortion of your shape. This is why the default behavior of the DRC blunts angles less than 45° before shrinking or bloating.

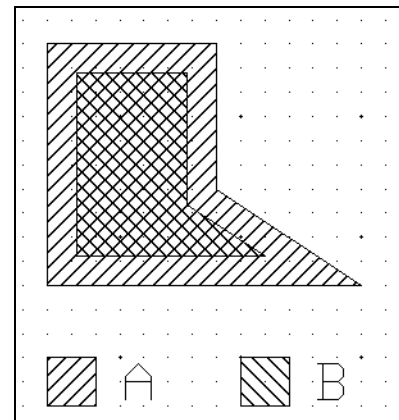


Figure 196: B = SHRINK (A, 1.2)

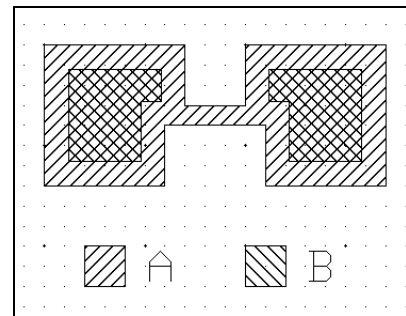


Figure 197: A single polygon on layer A becomes two polygons on layer B after shrinking.

If you are using SHRINK on polygons with angular notches, you should refer to the BLOAT_ANGLE rule on page 191 for important information on the effects that acute angles can have on this rule.

One common use of the SHRINK rule is to combine it with a BLOAT rule to remove all small polygons on a given layer. This can be used to classify irregular shapes, like wires, by size.

Example:

```
M1_SHRINK = SHRINK(M1_IN, 2.5)  
M1_OVER_5 = BLOAT (M1_SHRINK, 2.5)  
M1_OTHER = M1_IN AND NOT M1_OVER_5
```

See a more complete example of this process on page 65.

This set of rules will create shapes on M1_OVER_5 for all shapes on M1_IN that are wider than 5 units. All other shapes on M1_IN will be copied to layer M1_OTHER. This operation can have unfortunate side effects. Polygons of varying width can be distorted. Also, the shrink operation can distort the slope of sides that are not at 90° or 45° because the vertices of such sides after the shrink are often not on grid. The bloat operation then magnifies the problem. Look carefully at the layers created before you rely on them for design rules checking.

If the M1_IN layer contains acute angles, you should add pair of BLOAT_ANGLE rules around the BLOAT rule to prevent the acute angles from being cut by that rule. See page 191.

The shrink and bloat operations can also be relatively expensive in terms of processing time due to panel processing. If the layer you need to classify by size contains only rectangles or simple polygons, we suggest that you look at the BOUNDS or IS_BOX rules instead.

When the DRC is processing the result layer dangerously, the shrink rule may process shapes somewhat differently than you would expect. See page 135 for an example.

SNAP

Relocate vertices on resolution grid

$result_layer = \text{SNAP} (layer1, grid_resolution)$

Refer to page 79
for more details
on vertex
resolution.

This rule is used to reposition the vertices of all polygons on *layer1* so that they lie on the grid defined by *grid_resolution*. Vertices that are already on this grid are copied to *result_layer* unchanged. Off-grid vertices that this rule modifies will not be counted as errors. Shapes collapsed to zero width or height are eliminated without warning.

Define *grid_resolution* as a positive real number of units of the ICED™ cell.

Example:

B = SNAP (A, 1)

This rule will copy all polygons from layer A to layer B and relocate all vertices so that all coordinates lie on an integer grid. When this rule is executed on the shape on layer A in Figure 198, the shape shown on layer B is created.

Note that the slope of the shape on layer A has not been preserved. Shapes with sides at skewed angles will often be distorted when their vertices are relocated on grid.

To preserve the slope of 45 sides, you can use the SNAP45 rule.

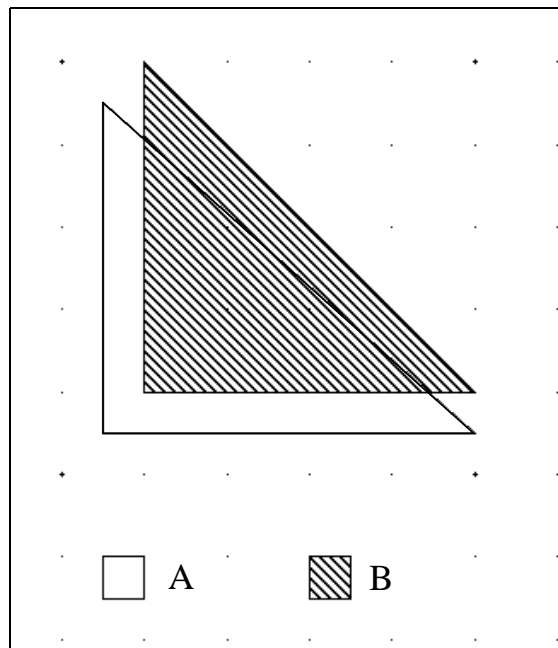


Figure 198: Polygon on A with off-grid vertices and polygon on B snapped to grid.

If *layer1* is not defined as an input layer, *result_layer* can be the same layer as *layer1*. This will replace layer1 with polygons with on-grid vertices.

Example: **A = SNAP (A, .05)**

In this example, layer A will be replaced with polygons with all vertices snapped to a grid with a .05 user unit resolution.

Also see the
SNAP45 and
OFF_GRID
rules.

Note On Touching Shapes

Remember that the DRC merges all touching shapes before verifying whether or not coordinates need to be snapped to grid. If two shapes on *layer1* share an edge on an off-grid coordinate, but the merged shape has no edges with off-grid coordinates, the shapes **will not** have their coordinates snapped to grid. See the overview of grid resolution issues on page 79 for details.

Preventing Off-Grid Coordinates

See the CUT_RESOLUTION rule to prevent off grid coordinates from being created by the DRC on generated layers at panel boundaries.

SNAP45 *Relocate vertices on resolution grid preserving slope of 45° angles*

result_layer = **SNAP45** (*layer1*, *grid_resolution*)

Refer to page 79 for more details on vertex resolution.

This rule is used to reposition the vertices of all polygons on *layer1* so that they lie on the grid defined by *grid_resolution*. Specify *grid_resolution* as a positive real number of user units of the ICED™ cell. This rule differs from the SNAP rule in that the slope of sides at a 45° angle will be preserved.

When a side with an off-grid vertex is at 45°, the **lower** vertex will be shifted in a manner that preserves the 45° slope. This may introduce a new vertex that adds a ledge or cuts off a corner.

Example:

C = A AND B
D = SNAP(C, 1)
D45 = SNAP45(C, 1)

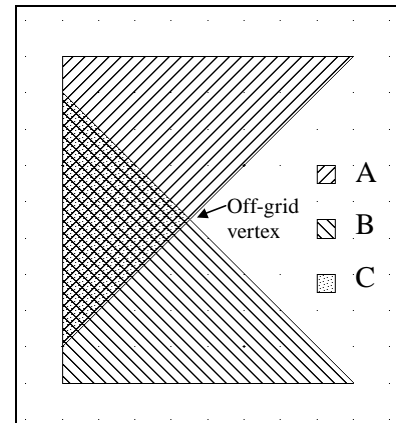


Figure 199: Intersection causes off-grid vertex.

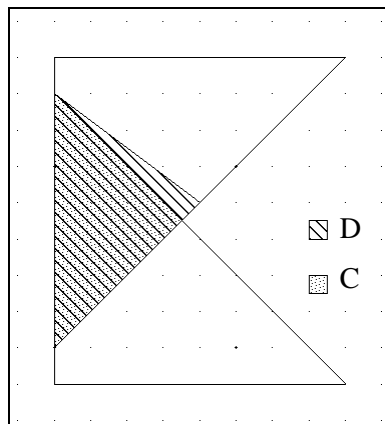


Figure 200: SNAP rule causes side to change slope.

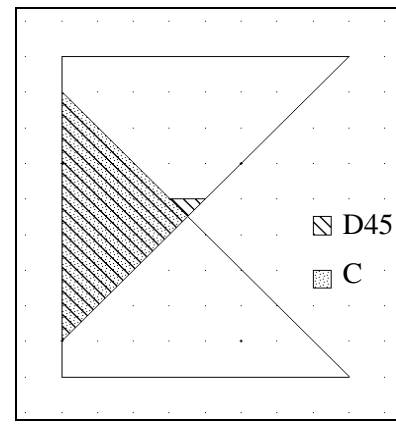


Figure 201: SNAP45 rule preserves 45° slope by adding ledge.

When the above set of rules is run on the shapes in Figure 199, the intersection causes the shape on layer C to have an off-grid vertex. When the SNAP rule is used to force the vertex onto an integer grid, the slope of one skewed side is no longer 45°. When the SNAP45 rule is used instead, a new vertex is added which maintains the 45° slope by adding a small ledge to the polygon.

Vertices that are already on the new grid are copied to *result_layer* unchanged. Off-grid vertices that this rule modifies will not be counted as errors. Shapes collapsed to zero width or height are eliminated without warning.

Also see the
SNAP and
OFF_GRID
rules.

There are some extreme cases where the SNAP45 rule is unable to snap a vertex to the required grid and preserve the 45° slope. When this happens, the DRC will snap the vertex to a point on a *grid_resolution/2* grid and issue a warning message in the log file.

Note On Touching Shapes

Remember that the DRC merges all touching shapes before verifying whether or not coordinates need to be snapped to grid. If two shapes on *layer1* share an edge on an off-grid coordinate, but the merged shape has no edges with off-grid coordinates, the shapes **will not** have their coordinates snapped to grid. See the overview of grid resolution issues on page 79 for details.

Preventing Off-Grid Coordinates

See the CUT_RESOLUTION rule to prevent off grid coordinates from being created by the DRC on generated layers at panel boundaries.

STAMP

Electrically connect poor conductors

STAMP *layer1* BY *stamping_layer* MULTI = *error_layer1* [NONE = *error_layer2*]

The DRC uses the electrical connections defined by this rule and the CONNECT rule to determine if shapes are electrically connected when executing some MIN_SPACING rules.

The STAMP rule is used to form electrical connections to layers that are poor conductors. Shapes on *layer1* that touch a shape on *stamping_layer* will be assigned the node number of the shape on *stamping_layer*. However, even when the shape on *layer1* touches other nodes on the *stamping_layer*, the DRC will not assign the node number of the *layer1* shape to shapes on the *stamping_layer*.

In other words, *layer1* is treated as a non-conductive material which can be "stamped" with a node number, but it cannot "stamp" any conductive layers. Electrical connections on the *stamping_layer* do not pass through *layer1*.

Opens that were not found because poor conductors were used as ordinary conductive layers by circuit recognition programs have caused chips to fail. This class of error is easily overlooked, but easy to verify with the DRC. In addition, you can verify that every well in your design is connected before beginning the circuit recognition process.

The Advanced Tutorial covers the use of the STAMP rule. See page 411.

The STAMP rule can be used to verify that every shape on *layer1* is electrically connected to **exactly** one node. Any shapes on *layer1*, which touch more than one node on the *stamping_layer* will be copied to *error_layer1*. All shapes on *layer1* that do not connect to any nodes on *stamping_layer* can be copied to *error_layer2* by using the optional NONE keyword. Shapes on both layers are added to the error count automatically.

Example: **OUTPUT ERROR LAYER 112 OVER_STAMPED_WELL**
STAMP WELL BY PDIFF MULTI= OVER_STAMPED_WELL

See page 116
for a more
detailed
explanation of
this example.

You must use
CONNECT
rules to form
electrical nets
from good
conductors to
verify a poor
conductor layer
with this rule.

We can demonstrate the importance of verifying well connections with Figure 202. Let us assume that the GND wire on the right connects to the metal GND bus and from there to a pad on the chip. However, the GND wire on the left does not connect to the bus. You meant to connect these two wires, but a gap exists by accident.

When the STAMP rule above is executed on the shapes in Figure 202, the WELL shape will be copied to the OVER_STAMPED_WELL since it will be stamped by two different nodes on layer PDIFF.

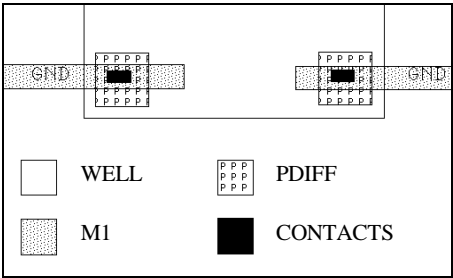


Figure 202: Open on GND node that connects only through WELL layer.

STOP_ON_MAX_COUNT *Halt DRC on maximum number of errors*

STOP_ON_MAX_COUNT

Use the
MAX_COUNT
rule to change
the maximum
error count.

By default, the DRC will warn you by posting a message on the screen when a maximum error count is reached. The default maximum error count is 1000. If you prefer that the DRC halt execution rather than just post a warning message, add the STOP_ON_MAX_COUNT rule to the rule set.

Before the DRC completes, it will close files properly allowing you to use the log and command files to troubleshoot the errors already found. The log file will contain a warning similar to the following near the end before the error summary information.:

```
**WARNING*****WARNING*****WARNING*****WARNING  
**Error count=1000 any further errors not reported. **
```

To understand why the DRC warns you when a maximum error count is reached, imagine a chip with 10,000 copies of a cell. If a small change to this cell causes a single error, there will be at least 10,000 error marks created for what you would consider a single error. Other error marks will be easily overlooked. The error would be caught just as well if the DRC stopped after the first 1000 errors, and the run time and output files would be much smaller. It is much more efficient to find and fix the single error in a shorter run, and then other errors will be easily seen in your next run.

TOUCHING

Find touching shapes on different layers

$result_layer = layer1 [NOT]^{17} \text{ TOUCHING } [n1 [:n2]] layer2 [NOT = result_layer2]^{17}$

See also the OVERLAPPING rule.

This rule is used to classify polygons on *layer1* based on whether or not they touch polygons on *layer2* along a finite line or area. Polygons touching only at a point, as shown in Figure 203, are **not** considered to be touching.

For shapes on *result_layer* to be considered errors, add the ERROR keyword to the OUTPUT LAYER rule that defines it.

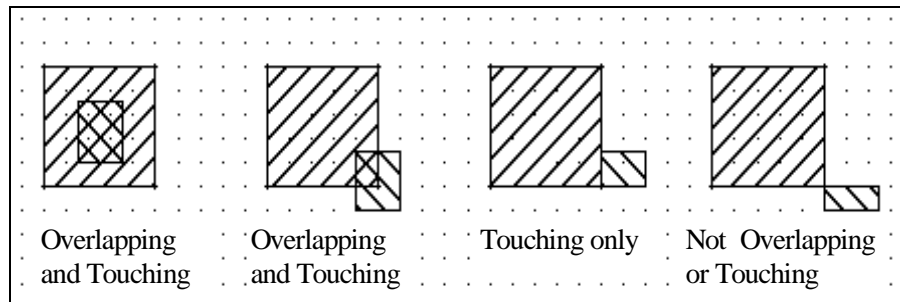


Figure 203: Differences between the OVERLAPPING and TOUCHING rules.

Example: **C = A TOUCHING B**

In this example, layer C will contain all polygons on A which touch at least one polygon on layer B

Only one optional NOT keyword can be used in the TOUCHING rule.

Example: **D = A NOT TOUCHING B**

In this case, layer D will contain all polygons on layer A which do not touch any polygons on layer B.

¹⁷ Only one optional NOT keyword is allowed in a single rule.

Example: **C = A TOUCHING B NOT = D**

Layer C will contain all polygons on layer A that touch at least one polygon on layer B. Layer D will contain all remaining shapes on layer A, i.e. all shapes not touching layer B.

This rule can be very useful to find shapes that are not completely covered by another layer. For example, if all shapes on layer A should be completely covered by shapes on layer B you may be tempted to write a simple Boolean rule to test for violations as in the following rule.

Example: **ERR = A AND NOT B**

However, the rule above will not mark problems like the one shown in Figure 204. If shapes on layer A must be completely enclosed by layer B, use a touching rule similar the following example to find violations.

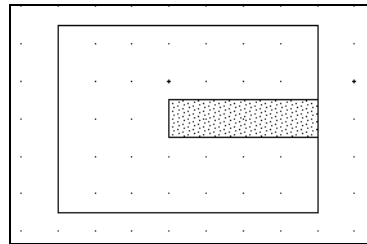


Figure 204: Incomplete enclosure.

Example: **NOT_B = NOT B**
ERR = A TOUCHING NOT B

The optional *n1* and *n2* parameters can be used to specify how many polygons on *layer2* the polygons on *layer1* must touch. Use *n1* alone to specify an exact number. Use both *n1* and *n2* to specify a range.

Example: **C = A TOUCHING 2 B**

In this case, layer C will contain all polygons on layer A that touch exactly two polygons on layer B.

Example: **C = A TOUCHING 2:4 B**

When you use this rule, layer C will contain all polygons on layer A which touch exactly two, three, or four polygons on layer B.

WARN_ACUTE

Assign layer number for acute angle warning marks

WARN_ACUTE = *layer_number*

See the MIN_ANGLE and MAX_ANGLE rules to find acute angles on a specific layer.

If you want to suppress the identification of acute angles entirely, use the NO_WARN_ACUTE rule instead of this rule.

The DRC can create shapes with acute angles when sides at an angle cross a panel boundary, or when shapes with holes or more than 199 vertices are output. Acute angles are frequently a problem for mask processing software. Whenever you create mask layers with the DRC, you should identify possible problems with specific shapes that should be fixed in the layout editor. See page 76.

Since this is so important, the DRC now automatically identifies acute angles (angles sharper than 90°) on all polygon output layers, marks the angles with error wires on a special output layer (layer number 99 by default), and lists them in the log file. Both acute angle protrusions and acute angle notches will be marked.

A warning will be added to the log file for each acute angle, however they will not be added to the main error count. The summary in the console messages and log file near the main error count will mention the acute angles.

If you want to change the layer number used to mark acute angles, add this rule to your rule set. The acute angles will be marked with wire shapes on *layer_number* in the main DRC command file.

Example:

WARN_ACUTE=101

When this rule is present anywhere in your rule set, at the end of the DRC run sides of shapes on **all** output layers that meet at an acute angle will be marked with wires on layer number 101 rather than the default layer number 99. Warnings will be printed in the log file for each acute angle similar to:

An acute angle was formed on output at (55, 15)

DRC Rules Syntax: WARN_ACUTE

If you want to suppress the creation of the wire shapes, but still want to count the acute angles and print the log file warnings, use this rule with a layer number of 0.

Example: **WARN_ACUTE=0**

See examples of
fixing acute
angles on pages
77 and 423.

Whenever you use layer number 0 as an output layer in a DRC rule, no output shapes are actually created in the command file. The use of layer 0 in this WARN_ACUTE rule means that a warning will still be printed in the log file for each acute angle, but no wires will be created to mark each angle in the command file.

If you want to suppress the identification of acute angles entirely, use the NO_WARN_ACUTE rule instead of this rule.

A non-zero *layer_number* defined with this rule is automatically added to the list of output layers. You do not need to define it with an OUTPUT LAYER rule.

WIRE_WIDTH

Set error wire width for all error layers

WIRE_WIDTH = *error_wire_width*

The WIRE_WIDTH option on the DRC command line will override any value set with this option. See page 355.

Use this rule to set the width used for all error wires created on all error layers.

When this rule is not used, or if you use the rule WIRE_WIDTH=0, when the DRC command file creates the error wires in the cell, they will be created using the default width of each error layer. These default widths are set in the ICED™ cell using the layout editor's LAYER command.

If you do not customize the width of error layers in the cell **before** executing the DRC command file, then the error wires can be difficult to see because they are often as wide as the shapes whose edges they are supposed to mark.

When you use this rule to set a non-zero width, all error wires are created with the specified width. Specify a width of around 10% to 20% of the average width of shapes in your layout. This will clearly mark edges of shapes.

Example: **WIRE_WIDTH=.2**

Adding this rule to the rule set will create all error wires with a width of .2.

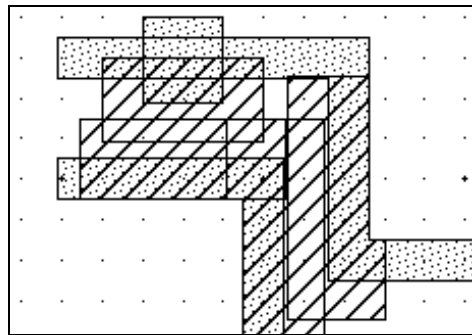


Figure 205: Confusing error wires of width=2.

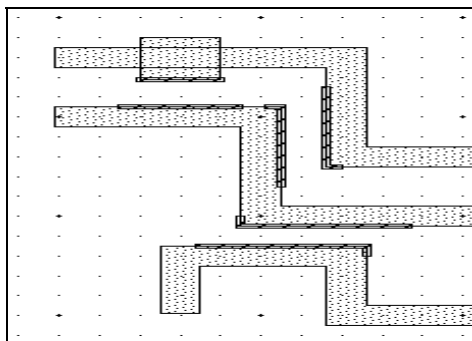


Figure 206: More distinct error wires of width=0.2.

XOR

Boolean exclusive OR

result_layer = [NOT] *layer1* **XOR** [NOT] *layer2*

XOR stands for "exclusive or". Use the XOR rule to create the union of the two layers and then subtract their intersection.

Example: **C = A XOR B**

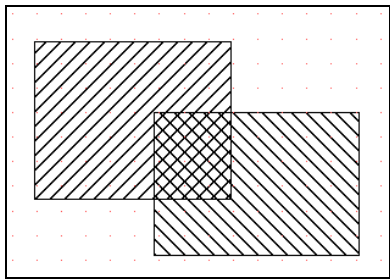


Figure 207: Polygons on layers A and B

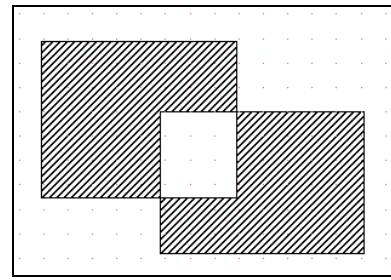


Figure 208: C = A XOR B

The optional NOT keywords work in exactly the same manner as they do in the AND rule.

Running the DRC

Running the DRC

Running the DRC involves several steps.

Write the DRC rules that define layer manipulation and design rule verification in any ASCII text editor.

See page 12 for a graphical representation of these steps.

Compile the rule set with the DRC rules compiler, D3RUL-NT.EXE¹⁸.

Create the binary layout data file for the DRC from your ICED™ cell by using the DRC command in the ICED™ layout editor.

Execute DRC3-NT.EXE¹⁹ using the compiled rules file and the binary layout data file.

Look at the results in the ICED™ layout editor by importing the shapes in the command file created by DRC3-NT.EXE.

A few hints on using the DRC command in the layout editor are provided on page 16.

The DRC command for the ICED™ layout editor is completely described in the layout editor reference manual. To export your entire design, you can simply type "DRC" on the command line. The binary layout data file will be created as "*cell_name*.POK".

We have already covered how to write the rule set. Next, we will describe how to run the rules compiler and finally how to run the DRC program. Tips on how to execute the command file to import the results are covered last.

¹⁸ The executable file for released versions for Windows is D3RUL-NT.EXE. The executable file for Beta Windows versions is named D3RU-NTX.EXE.

¹⁹ The executable file name for released versions for Windows is DRC3-NT.EXE. The executable file name for beta Windows versions is DRC3-NTX.EXE.

DRC Rules Compilation

Rules Compiler Command Line Syntax

```
[prog_path\]D3RUL-NT20 [rule_path\]rule_file_name ...  
... [ BB_FILE=output_file_spec ] ...  
... [ HOG=mem_megabytes ] ...  
... [ USE=mem_kilobytes ] ...  
... [ PAUSE=(ALWAYS | CRASH | NEVER)] ...  
... [ SCRATCH_DIR=scratch_path1 [; ... scratch_path5 ] ]
```

Type the compiler command line in the console window opened by the ICED icon on your desktop.

The *rule_file_name* parameter is the only required parameter on the rules compiler command line. It is the name of the ASCII file containing your rule set. A file extension of ".RUL" will be added to the file name if you do not supply the file extension in *rule_file_name*.

The DRC installation defaults to placing all executable files in the same directory as the ICED.EXE file. This directory is added to the PATH environment variable in the console window opened by the ICED icon on your desktop. When you use this console window to execute the DRC programs, you do not need to supply *prog_path* on the command line.

²⁰ The executable file for released versions for Windows is D3RUL-NT.EXE. The executable file for Beta Windows versions is named D3RU-NTX.EXE.

Running the DRC: Rules Compilation

Example: **CD RULEDIR**
 D3RUL-NT MYRULES

To see a list of the current DOS environment variables, use the DOS command SET.

The CD DOS command changes the current directory. The next command line will execute D3RUL-NT.EXE on the file MYRULES.RUL. Since no *rule_path* is supplied, this rules file must exist in the current directory, RULESDIR. The executable file D3RUL-NT.EXE must be in a directory defined in the DOS environment variable PATH. This search path is initialized correctly for you by the ICED icon.

By default, the compiler will create the compiled rule file with the same name as the input rule set except that the file extension will be ".BB". By default, this file will be created in the same directory as the input rule file. In the example above, the compiled rules file will have the name MYRULES.BB. It will be created in the current directory.

Output Redirection

You can use the `BB_FILE=output_file_spec` option to specify a different name and/or directory location for the compiled rules file. If you specify a new directory location, the directory must already exist.

If *output_file_spec* ends in a '\', then the rules compiler assumes that you are specifying only a directory location. The rules file will use the default name as described above, but it will be stored in the directory specified.

Example: **D3RUL-NT MYRULES.RUL BB_FILE=\XDIR**

The compiler command line above will compile the file MYRULES.RUL in the current directory and store MYRULES.BB in the XDIR directory at the root of the current drive. The MYRULES.RLO file (the compiler log file) will remain in the same directory as the source rules file MYRULES.RUL.

Example: **D3RUL-NT MYRULES.RUL BB_FILE=SUBDIR**

The compiler command line above will create MYRULES.BB in the SUBDIR subdirectory of the current directory.

Example: **D3RUL-NT MYRULES.RUL BB_FILE=E:\XDIR\NEWNAME**

The compiler command line above will store the compiled rules in the file NEWNAME.BB in the E:\XDIR directory.

Example: **D3RUL-NT MYRULES.RUL BB_FILE= NEWNAME**

The compiler command line above will store the compiled rules in the file NEWNAME.BB in the current directory.

Memory Options

You can create a DOS batch file containing the rules compiler command line so that you do not need to type it each time. See page 359.

The optional [*USE=mem_kilobytes*] or [*HOG=mem_megabytes*] parameter is used to restrict the amount of memory the rules compiler will use. Use only one or the other since they are mutually exclusive. When neither keyword is used in the command line, the rules compiler will use 10 Megabytes (approximately 10,000 Kilobytes) of memory. The rules compiler does not usually require a larger amount of memory, so you can usually avoid the use of either keyword. (Either keyword can be used on the DRC program command line, where they are more useful.)

If you notice a long delay when you execute the rules compiler, the problem may be that the rules compiler is initializing much more memory than it needs. Try using *USE=2000* to prevent the compiler from initializing too much memory on your system.

Running the DRC: Rules Compilation

Example: **D3RUL-NT SUBDIR\MYRULES.RUL USE=2000**

If your operating system is pre-parsing the command line and replacing the '=' with a blank, use '#' instead. See page 333.

This compiler command line will limit the program to approximately 2 Megabytes of memory. If you have a short rule set, this will be plenty of memory. This rule file specification means that the compiler will use the MYRULES.RUL file in the SUBDIR subdirectory of the current directory. The compiled rules file, MYRULES.BB, will be created in this directory as well as the compiler log file, MYRULES.RLO.

Batch Console Window Control

The console window will not close automatically when you type the command line in an open console window.

The following option is useful only when executing the compiler from a batch file. When you execute the compiler from a batch file without opening a console window explicitly, the compiler executes in a temporary console window. In this case, the console window may close automatically without giving the user time to see the final messages posted as the compiler terminates. Once the console window is closed, the console messages are gone forever. If the DRC crashes, it is very important to review these console messages to avoid repeating a wasted run.

Some users create a batch file with the DOS PAUSE command after the compiler command line to ensure that the console messages remain on the screen until they can be viewed. With this version of the DRC, the extra PAUSE command can be omitted. If you prefer to hold the window open until the <Return> key is pressed, simply add a PAUSE option to the compiler command line.

- PAUSE=ALWAYS** This will always pause after termination until the <Return> key is pressed.
- PAUSE=CRASH** This option pauses the compiler after termination only when the program crashes. If the program terminates normally, no PAUSE is executed and no keystroke is required to close the console window.
- PAUSE=NEVER** (This is the default if no PAUSE option is used on the command line.) The window closes without an extra keystroke. If the console window will not close automatically when the DRC terminates, this is the best option.

Scratch Directories

If the `SCRATCH_DIR` keyword is not used, a single scratch file, `$D3RVIRT.000`, will be created in the current directory. If the compiler completes successfully, this file will be deleted.

There are two cases where it is important to use the `SCRATCH_DIR` option to set the scratch directory explicitly.

If you running the DRC rules compiler on a network, several users can use the program at the same time. If they share a scratch directory, they will corrupt each other's scratch files. **When on a network, each user should have his\her own scratch directory.**

If the current drive or partition has limited space, you should specify at least one scratch directory on a drive with plenty of free space.

The scratch file for the rules compiler is usually very small, so the second reason above is rarely a concern. However, the scratch file for the DRC program can grow very large. The `SCRATCH_DIR` command line option is covered in more detail in the discussion of the DRC command line options. See page 341.

Running the DRC: Rules Compilation

Example: **D3RUL-NT D:\ICED\VERIF.RUL SCR=E:\DRCTMP**

This command line will compile the rules in the file D:\ICED\VERIF.RUL. Note that the SCRATCH_DIR keyword can be abbreviated to SCR. The DRCTMP directory on the E: drive will be used to store the temporary scratch file. This directory must already exist or the rules compiler will fail with an error message.

Terminating the Rules Compiler

There are two ways to force the rules compiler to terminate before it completes normally.

<Esc> Pressing this key will halt the rules compiler after it completes the current operation. The compiler will then close all open files and delete the scratch file(s). This may take a few moments, so you should be patient and wait for the compiler to complete these tasks.

<Ctrl><C> Pressing both of these keys simultaneously will bring the rules compiler to an immediate halt. Files will not be closed properly and the scratch file(s) will not be deleted.

If you use <Ctrl><C> to terminate the compiler, or if the compiler crashes, you should delete the scratch file(s) yourself. The scratch file is created with the name \$D3RVIRT.000. You may also want to run the DOS utility SCANDISK (or equivalent programs available from other vendors) to find lost chains on the disk which may be left behind because files were not closed properly.

Rules Compiler Output Files

To suppress the warning prompt when the source rules file is not present, add the NO_RUL keyword to the DRC command line. See page 349.

The *rule_file_name*.TAG file generated by the rules compiler is intended for use by the interactive DRC features of ICED™.

By default, the rules compiler will create the compiled rules file with the name *rule_file_name*.BB. If a file with the name *rule_file_name*.BB already exists, it will be overwritten. This file will be created by default in the same directory as the source rules file. (The BB_FILE option can be used to change the name and/or location of the compiled rules file.) This compiled rules file will be used by the DRC program.

You should leave the source rules file in its original location after you have compiled it. The location and time/date stamp of the source rules file is stored in the compiled rules file. The DRC will search for the source rules file to insure that the current source rules file has the same time/date stamp as the one used to create the compiled rules file. This prevents a wasted DRC run when you edit the rules file but then forget to compile it. If the DRC cannot locate the source rules file, it will issue a warning prompt and you must reply to proceed.

The rules compiler will create a log file with the name *rule_file_name*.RLO. This file will always be created in the same directory as the source rules file. If a file with the name *rule_file_name*.RLO already exists, it will be renamed to *rule_file_name*.RL1.

The rules compiler log file begins with a block of comments that include the version number of the compiler. An echo of the source rules file comes next. Any warnings or errors listed in the console messages about the rule set will be listed here as well.

If the compiler finds a syntax error as it parses the source rules file, it will stop reading the file and print an error message after the line with the error. The parameter or keyword with the problem will be indicated with carats ("<>"). Only one error will be found per compilation.

Warnings may be scattered through the log. All warnings will be prefixed with the string "***WARNING". Some of the warnings you may see are listed below.

Message	Cause
Scratch layer <i>xxx</i> , set on line <i>n</i> , is never used. Action will be deleted: ...	<p>This warning occurs when you have included a rule that creates shapes on a scratch layer, but no succeeding rule uses that layer. The processing to create the scratch layer is unnecessary, so the rules compiler deletes the rule entirely. This situation may occur when you modify a rule set by removing a rule that used the scratch layer. The DRC will then optimize your rule set by removing the rules that create the layer. In this case you can choose to ignore the warnings, or go back and comment out the indicated rule(s).</p> <p>However, if you wanted to look at the shapes on that scratch layer, you should change the line that defined the layer to an OUTPUT LAYER rule instead of a SCRATCH LAYER rule.</p>
Layer number <i>n</i> is also an input layer.	<p>When this message is issued, you have used the same layer number as both an input layer and an output layer. If you add shapes on the output layer to your design cell you will be modifying a design layer.</p> <p>You can define the layer with a MODIFY LAYER rule to avoid the warning.</p>

Figure 209: Two of the DRC rules compiler warnings

If the source rules file contains no syntax errors, the log file will continue with a summary of the layers used in the file. The DRC layer name, ICED™ layer number, the rules file line number that defined the layer, and the layer type (INPUT, OUTPUT, or SCRATCH) will be listed for each layer.

Unused input layers are still checked for bad polygons unless the NO_CHECK-_INPUT rule is used.

Layers that are defined, but are not used in the rule set, will not be listed in this list of layers. They are removed automatically from the rule set by the compiler.

Next, the rules log will list all constants created by the CONST rule. (See page 203.)

The
LIST_RULES
option on the
DRC command
line will add a
listing of rules
to the DRC log
file.

The log will then list the rules exactly as they will be executed by the DRC. The operation number (or action number) for each rule is listed first. The rules are grouped together in passes. Each pass requires each shape in the DRC database to be interrogated by the DRC. The more passes, the longer the DRC run.

The order in which rules are executed may not be the order in which the rules are written. The rules compiler may change the order to minimize the number of DRC passes. No change made by the compiler should affect how the layers are processed.

One example of
a rule generated
by the compiler,
is a CONNECT
rule added to
insure that
shapes that
cross panel
boundaries will
be handled
correctly.

Each DRC layer name which is an input or output layer will be followed by the ICED™ layer number enclosed in square brackets ("[]"). The line number in the source rules file is indicated on the line after the rule enclosed in parentheses ("()"). If the compiler has generated the rule, the word "Generated" will be used instead of the line number. Some additional information may be provided with the rule, such as the bloat angle in effect for BLOAT or SHRINK rules.

If you have defined electrical connections through the use of CONNECT rules, some details on these electrical connections are listed next. The number of groups formed from the electrical connections, and the layers in each group, will be listed. If your log indicates more than one group, you may have omitted a CONNECT rule from your rule set. See page 110 for more details.

Layers in the rule set that are not electrically connected to any other layer are listed under the heading "Unconnected layers". This list may contain intermediate layers, or layers that are never used. However, you may want to browse this list to insure that none of the layers that you assume are electrically connected are included in the list.

The final line in the log file from a successful compilation will always be the word "Done".

Running the DRC

DRC Command Line Syntax

<code>[prog_path\]DRC3-NT²¹ [rule_path\]rule_file_name ...</code>	} File Parameters Page 334
... <code>[layout_path\]layout_file_name ...</code>	
... <code>[output_path\]output_file_base_name ...</code>	
... <code>[SECOND_CELL=layout_file_name2] ...</code>	
... <code>[@opt_file] ...</code>	— Input Redirection Page 334
... <code>[QUICK_PASS] ...</code>	} Algorithm Options Page 337
... <code>[ALLOW_QUICK] ...</code>	
... <code>[SLOW] ...</code>	
... <code>[QUICK_SPACING] ...</code>	
... <code>[USE=mem_kilobytes] ...</code>	} Memory Options Page 339
... <code>[HOG=mem_megabytes] ...</code>	
... <code>[NO_VIRTUAL_MEMORY] ...</code>	
... <code>[MAIN_MEMORY=main/total_ratio] ...</code>	
... <code>[MAIN_USE=main_kilobytes] ...</code>	
... <code>[MAIN_HOG=main_megabytes] ...</code>	
... <code>[SCRATCH_DIR=scratch_path1 [; ... scratch_path5]] ...</code>	
... <code>[FILESIZE=scratch_megabytes] ...</code>	
... <code>[SHORTRUN] ...</code>	} Screen Display Options Page 343
... <code>[LONGCASE] ...</code>	
... <code>[DISPLAY_OPERATIONS=min_refresh_seconds] ...</code>	
... <code>[NO_FLASH_PANELS=flash_limit] ...</code>	
... <code>[PAUSE=(ALWAYS CRASH <u>NEVER</u>)] ...</code>	

(continued on next page)

²¹ DRC3-NT.EXE is the name used for released Windows versions of the program. Other versions use different names. See page 331.

Running the DRC: Command Line Syntax

... [LAYERS=(<i>layer_number1</i> [..., <i>layer_numbern</i>])] ...	} Rules File and Log File Options Page 346
... [DO=(<i>rule_spec1</i> [..., <i>rule_specn</i>])] ...	
... [SHOW_BORDER] ...	
... [BORDER= <i>border_dimension</i>] ...	
... [NO_RUL] ...	
... [LIST_RULES] ...	
... [SHOW_SCALES] ...	
... [LEFT= <i>left_x_coordinate</i>] ...	} Design Area Options Page 351
... [RIGHT = <i>right_x_coordinate</i>] ...	
... [TOP= <i>top_y_coordinate</i>] ...	
... [BOTTOM= <i>bottom_y_coordinate</i>] ...	
... [FLATTEN] ...	} Cell Hierarchy Options Page 352
... [NO_FLATTEN] ...	
... [CFLATTEN= <i>component_count</i>] ...	
... [NFLATTEN= <i>use_count</i>] ...	
... [HIERARCHICAL=" <i>suffix_string</i> "] ...	
... [WIRE_WIDTH= <i>error_wire_width</i>] ...	} Command File Options Page 356
... [START_CMD=" <i>st_cmdstring</i> "] ...	
... [END_CMD=" <i>end_cmdstring</i> "] ...	
... [OBSOLETE] ...	
... [MACROS=NONE] ...	
... [PANEL_X= <i>panel_x_dimension</i>] ...	} Panel Size Options Page 358
... [PANEL_Y= <i>panel_y_dimension</i>] ...	
... [PANEL_A= <i>panel_area</i>] ...	
... [PANEL_X_BY_Y= <i>panel_ratio</i>] ...	

The command line used to execute the program is reported in the DRC log file.

The DRC command line is typed at the DOS prompt, or in a batch file, outside of the ICED™ layout editor. The first two input files, *rule_file_name* and *layout_file_name*, must already be prepared before you execute the program. All three required file parameters and each optional parameter are described in detail on the following pages.

If the directory where DRC3-NT.EXE is installed is included in the DOS environment variable PATH, or if this directory is the current directory, the *prog_path* parameter is not required. (The console window opened by the ICED icon on your desktop adds the main installation directory to PATH automatically.)

Name of the Program

Type the DRC command line in the console window opened by the ICED icon on your desktop, or use a batch file.

The name of the program is different for different versions of the DRC. The different versions have different file names, so you can keep more than one version on your machine without risking overwriting what you are currently using.

DRC3-NT.EXE is the name of the released Windows executable file. In this manual, all examples use this version name in the command line. (DRC3-NT.EXE is shortened to DRC3-NT in the example command lines since the operating system will translate the name of the executable file to DRC3-NT.EXE.)

If you use the DOS version of the program, or a beta version, replace the string “DRC3-NT” in the example command lines with the appropriate entry from the “Command Line String” column in the table below.

Version	Executable file	Command Line String
DOS released versions	DRC3.EXE.	DRC3
Windows released versions	DRC3-NT.EXE.	DRC3-NT
Windows beta versions	DRC3-NTX.EXE.	DRC3-NTX

Beta test versions of the DRC are frequently available on the IC Editors, Inc. web site. (www.iceditors.com). You can download a beta test version to your Q:\ICED directory to test new features without risking overwriting the version you are currently using in production. New features are tested in beta versions before they are reflected in the released versions. Remember that while a beta version may have more features, we call it beta **testing** for a reason.

Running the DRC: Command Line Syntax

You must use the equivalent version of the rules compiler to recompile the rules file before using a new version of the DRC.

Terminating the DRC

There are two ways to force the DRC to terminate before it completes normally. (These are the same methods used with the DRC rules compiler.)

<Esc> Pressing this key will halt the DRC after it completes the current operation. The DRC will then close all open files and delete the scratch file(s). This may take a few moments, so you should be patient and wait for the DRC to complete these tasks.

<Ctrl><C> Pressing both of these keys simultaneously will bring the DRC to an immediate halt. Files will not be closed properly and the scratch file(s) will not be deleted.

Read more
about the
scratch files on
page 341.

If you use <Ctrl><C> to terminate the DRC, or if the DRC crashes, you should delete the scratch file(s) yourself. The scratch file(s) are created with the name \$D3VIRT.000. You may also want to run the DOS utility SCANDISK (or equivalent programs available from other vendors) to find lost chains on the disk left behind because files were not closed properly.

Simultaneous DRC Runs

When using the DRC on networks or multitasking operating systems, do not launch multiple runs of the DRC program from the same directory. Scratch files or other temporary files in the current directory may collide. Simultaneous runs of the DRC should have no problems as long as they are started from different directories and use different scratch directories. (See the SCRATCH_DIR option on page 341.)

Command Line Options

The parameters on the DRC command line are read from left to right. If two conflicting parameters are encountered, the one on the right will be used without warning. Use blanks or commas to separate command line parameters. The underscores, '_', used in several of the optional keywords are included for readability only. You can type the keywords with or without the underscores.

The keywords can be abbreviated as long as you provide enough characters to make the keyword unambiguous. To abbreviate a keyword, drop characters from the end. Do not skip letters in the middle of the keyword. For example, DISPLAY_OPERATIONS can be abbreviated as DISP, but DISP_OP will cause a syntax error.

Using '#' in Place of '=' in Command Line Options

There are times when DRC command line options are parsed by the operating system in a way that replaces all '=' with a blank space. This will result in syntax errors by the time the DRC gets the command line. If you see syntax errors caused by this type of "pre-parsing", use the '#' character instead of '=' when typing the command line options. The '#' is never replaced by any operating system command line parser, and the DRC will translate it to an '=' when it parses the command line.

Example: **DRC3-NT MYRULES MYPOK DRCOUT USE#2000**

File Parameters

The *rule_file_name*, *layout_file_name*, and *output_file_base_name* parameters are the only required parameters on the DRC command line. (In some cases the QUICK_PASS or SLOW option is required. More on this later.)

The [*rule_path*]*rule_file_name* parameter supplies the name of the compiled rules file. This file must already have been created by the DRC rules compiler described beginning on page 319. If no file extension is supplied in *rule_file_name*, a file extension of .BB will be added to the file name before the DRC searches for the file.

If you specify *rule_path*\ with the name of the rules file, the DRC will search for the rules file only in that directory. When *rule_path*\ is not specified, the following directory paths will be searched in the order shown:

- 1) the current directory,
- 2) the directories in the environment variable DRC_PATH.

Environment variables are set at the console prompt or in a batch file (e.g. AUTOEXEC.BAT) with the DOS command SET.

The [*layout_path*]*layout_file_name* parameter must be the name of the binary layout file created by the DRC command in the ICED™ layout editor. A file extension of .POK will be added to the file name when you do not supply it on the command line. When you do not specify *layout_path*, the layout file must be in the current directory. The DRC will not search for layout files in directories set with the DRC_PATH environment variable.

See an example of creating a layout data file on page 16.

When you change the layout, ALWAYS recreate the layout data file with the DRC command in the layout editor BEFORE executing the DRC program. If you forget to recreate the file, the DRC will use the old layout data without warning!

The [*output_path*]*output_file_base_name* parameter supplies the base file name for most output files. The file extensions of the output files will vary. The extensions and file contents are described beginning on page 361.

Example: **DRC3-NT MYRULES XCHIP XCHIPOUT**

This command line will run the DRC with the input files MYRULES.BB and XCHIP.POK. The output command file and the log file will begin with the string "XCHIPOUT" in the current directory.

[SECOND_CELL=*layout_file_name2*]

When you use this optional parameter, the DRC can compare two layouts. The layer numbers in *layout_file_name2* will be shifted by 1000 so that the rule set can distinguish between layers in the first layout file and the second.

Example: **DRC3-NT CPM1 XCHIP COMP SECONDCELL=BACKUP\XCHIP**

The
HIERARCHICAL
command line
option is
incompatible
with this option.

For this example, let us assume that you have backed up the layout for the previous version of your design in the BACKUP subdirectory of the current directory. Now you want to compare the M1 layer in the two versions. The layer M1 in your design is layer number 1. The rule set for this comparison, CPM1.RUL, would be similar to the one below.

ALL_SAFE
INPUT LAYER 1 M1; 1001 M1_OLD;
OUTPUT LAYER 50 NOT_THE_SAME

NOT_THE_SAME = M1 XOR M1_OLD

When this rule set is compiled and used by the DRC with the command line given above, it will compare layer 1 in the two layouts and create all differences in the two layouts on layer 50 in the command file COMP.CMD.

Input Redirection

[*@opt_file*]

Another way to avoid repetitive typing of the DRC command line is to use a batch file to execute the program. See page 359.

Example:

The DRC has many optional parameters and the command line can get very lengthy. Since DOS commands are limited to 128 characters, you may not be able to add all of the options you need on the command line. To solve this problem, or just to save you from repetitive typing every time you run the DRC, you can use the *@opt_file* parameter to refer to a file which contains command line options, rather than typing all options at the DOS prompt.

DRC3-NT MYRULES XCHIP XCHIPOUT @DRCOPT.TXT SLOW

This DRC command line will cause the DRC to read the command line parameters in the file DRCOPT.TXT and execute them as though they were typed at the command line. Note that you can add other command line options after the *@opt_file* parameter. Options typed last on the command line override options in the options file.

!DRC options for xchip LEFT=348.8 RIGHT=1098.3 TOP=736.0 BOTTOM=-390.0 QUICK_PASS
--

Figure 210:
DRCOPT.TXT

The options file can be created with any ASCII text editor. Any end-of-line character is interpreted as a blank space. So you can type each command line option on it's own line rather than typing all options on a single line.

Comments in the options file are acceptable and will be ignored by the DRC. A comment is created as an exclamation mark (!) followed by text. All text from the exclamation mark to the end of the line is ignored.

You can use another *@opt_file* command line option in an option file. Option files may be nested up to ten levels deep.

Algorithm Options

[QUICK_PASS] or [SLOW]

The number of passes for the rule set is shown in the rules compiler log file. See page 327.

The QUICK_PASS and SLOW keywords are not truly optional. **Unless the rule set executes in a single pass, one of these options must be chosen.** They are mutually exclusive options.

If you use the HIERARCHICAL keyword in the command line (used to force the DRC to produce hierarchical output), the SLOW option is automatically invoked. If the rule set is multi-pass, and the HIERARCHICAL keyword is not used, you must specify one of these keywords on the command line. You will receive an immediate warning from the DRC if the algorithm choice has not been made.

The QUICK_PASS option may result in a warning prompt. To avoid the prompt, see the ALLOW_QUICK option covered on the next page.

The QUICK_PASS option will execute your rule set much more quickly at the expense of ignoring some of the more time-consuming operations. This option is intended to allow you to get DRC results faster on repeat runs when you have made minimal changes to the layout since verifying the design with the SLOW option on the DRC command line.

The QUICK_PASS algorithm will execute your rule set differently in the following ways:

Electrical connection rules and the /CONN or /~CONN options in MIN_SPACING rules are ignored. The CONNECT and STAMP rules in the rule set are not executed.

All BRIDGE, ISLANDS, MAX_SPACING, TOUCHING, and OVERLAPPING rules are skipped. Rules using layers generated by these rules are also skipped.

Shapes crossing panel borders are processed differently and may be misinterpreted in rare cases. (See page 129 for a more complete explanation.)

[ALLOW_QUICK]

When the QUICK_PASS option is used, some rules may be skipped. When rules will be skipped, by default the user is warned before execution with a warning prompt. The user will need to respond by typing a <Y> to proceed.

See also the ALLOW_QUICK rule on page 182.

If you want to avoid the warning prompt and the user interaction at run time, add ALLOW_QUICK to the command line **or** add the ALLOW_QUICK rule to the rule set. Either method results in the suppression of the warning prompt.

[QUICK_SPACING]

See page 100 for more details on the effects of this option.

The DRC has two algorithms for executing MIN_SPACING rules. This option is provided to force the DRC to use the quicker algorithm. The DRC will automatically use this algorithm if you are verifying the entire cell and your rule set does not contain MIN_SPACING rules using the /LENGTH keyword to force the DRC to discard errors shorter than a minimum length.

If you have used the design area options (LEFT=*left_x_coordinate*, etc.) on the DRC program command line, or if your rule set contains MIN_SPACING rules that use the /LENGTH option, adding the QUICK_SPACING keyword to the command line may cause errors to be missed. Do not use the QUICK_SPACING keyword on the final tests of your design.

Memory Options

[USE=*mem_kilobytes*] or [HOG=*mem_megabytes*]

You can use one of these parameters to define the maximum amount of memory the DRC is allowed to use. They are intended primarily for use on multitasking operating systems like Microsoft Windows. If you are running the DRC on a computer without a multitasking operating system, it is best to use the default memory parameters by not adding either option to the command line.

The default memory management refined with these options is incompatible with QEMM. See the NO_VIRTUAL_MEMORY option if you use QEMM.

When these parameters are not used, the DRC will allocate all available physical memory.

The USE and HOG options perform exactly the same function. The only difference is that the USE option specifies the amount of memory in Kilobytes, while HOG specifies the number in Megabytes. Use whichever option is more convenient.

If you see a long delay before the first DRC pass is executed, even for small designs, the DRC may be initializing much more memory than it needs. You can use a command line similar to the following to limit the amount of memory the DRC will initialize.

Example:

DRC3-NT MYRULES XCHIP XCHIPOUT USE=20000
DRC3-NT MYRULES XCHIP XCHIPOUT HOG=20

If the DRC is failing to run to completion with the memory available, try smaller panels. See page 118.

Either equivalent DRC command line will limit the DRC to around 20,000 Kilobytes (or about 20 Megabytes) of memory. This should be sufficient for small chips. If you have plenty of memory on your machine, and the other system demands are light, higher numbers for *mem_kilobytes* will allow the DRC to run faster.

You can set USE or HOG to a number higher than the amount of physical memory on your system. However, this means that the DRC will use your system's swap file for main memory and this may slow the DRC considerably.

[NO_VIRTUAL_MEMORY]

This command line option reverts the program to the older memory management method used in previous versions of the DRC.

The only situation we know of that requires this option is if you are using the QEMM memory manager in the Windows operating system. If you discover another situation that requires this option, please contact IC Editors.

[MAIN_MEMORY= *main/total_ratio*]

[MAIN_USE= *main_kilobytes*]

[MAIN_HOG= *main_megabytes*]

The DRC divides the total memory available to it into two portions: main memory (used for computations) and data storage (easier to swap to disk). By default, the DRC will divide the memory equally up to a limit of 128 Megabytes of main memory. (In other words, if you have 256 Megabytes or less available, half will be used for main memory, half will be used for data. If you have more than 256 Megabytes available, 128 Megabytes will be used for main memory, and the remainder will be used for data.)

If you want to change the division between main memory and data memory, you can use **one** of the three options below: (If you provide conflicting options, the least amount of main memory indicated will be allocated.)

MAIN_MEMORY	Specify ratio of main memory to total memory.
MAIN_USE	Specify main memory in Kilobytes
MAIN_HOG	Specify main memory in Megabytes

Example: **DRC3-NT MYRULES MYPOK DRCOUT HOG=10 MAIN_MEM=.6**

The HOG option limits the total amount of memory the DRC can use to 10 Megabytes. The DRC will allocate 6 Megabytes for main memory since the MAIN_MEMORY option (abbreviated to MAIN_MEM) is set to .6 rather than the default of .5.

The command line below will limit main memory to 6 Megabytes no matter what amount of memory is available on the system.

Example: **DRC3-NT MYRULES MYPOK DRCOUT MAIN_HOG=6**

The MAIN_USE option performs exactly the same function as the MAIN_HOG option except that you express the amount of memory in Kilobytes. The next example is almost identical to the previous example (give or take a few kilobytes).

Example: **DRC3-NT MYRULES MYPOK DRCOUT MAIN_USE=6000**

[SCRATCH_DIR=*scratch_path1* [; ... *scratch_path5*]]

Set the maximum size of all scratch files added together with the FILESIZE command line option (covered next).

This parameter specifies the directory (or directories) for the DRC scratch file (or files). **It is critical that a scratch directory is not shared between simultaneous executions of the DRC. This could be a problem if two or more users are executing the DRC over a network.**

If you do not use the SCRATCH_DIR command line option, the DRC will create a single scratch file with the name \$D3VIRT.000 in the current directory. This file can grow very large (more than 1 Gigabyte for large chips), so make sure that there is plenty of free space on the current drive.

See the next page to use this option to get around the 2 Gigabyte file size operating system limit for the scratch file.

You may want to use this option if you have limited space on the current drive and you want to make use of the space on other drives for scratch files. If you specify multiple directory paths, the DRC will create scratch files in all specified directories at the start of the run. Then if the DRC runs out of disk space while using the scratch file in the first directory, it will use the additional scratch files.

You can specify up to five directories. Each *scratch_path* directory should already exist. The maximum number of characters in all directory paths is 2047. The additional scratch file directories should usually be on other disk drives or partitions.

When you want to allow the DRC to use space on more than one drive, specify directories on different drives.

Example: **DRC3-NT MYRULES XCHIP XCHIP SCR=E:\DRCTEMP;D:\DRCTEMP**

Refer to page 166 for other information related to executing the DRC on very large designs.

This command line will result in scratch files created on the E: and D: drives in directories with the name DRCTEMP. Note that the SCRATCH_DIR keyword can be abbreviated to SCR. The directory paths are separated with semicolons. If the DRC completes successfully, all temporary scratch files will be deleted automatically at the end of the run.

Operating systems limit the maximum size of a single file to 2 Gigabytes, even on FAT32 partitions. If you have a single partition with more than 2 Gigabytes available and you expect to require a scratch file larger than 2 Gigabytes, you can get around the file size limit by specifying more than one directory on a single drive or partition. When the DRC has used 2 Gigabytes of scratch space in the first directory, it will begin using a second swap file in the next directory.

[FILESIZE=*scratch_megabytes*]

This optional parameter is used only when conserving memory is very important. It allows you to override the maximum size of all scratch files combined. The default is 2048 Megabytes multiplied by the number of directories in the SCRATCH_DIR option (covered above). If the default scratch directory is used, or if you have specified a single directory with the SCRATCH_DIR option, the default limit is approximately 2 Gigabytes.

Setting *scratch_megabytes* to a value smaller than 2048 will conserve memory. This is due to the fact that the DRC creates a virtual array page table in memory. The larger the maximum size of the scratch files, the larger this table must be.

There is no point to setting *scratch_megabytes* to a value larger than 2048. If you want to increase the scratch file size, modify the SCRATCH_DIR option (covered above) to use multiple directories and the default value of *scratch_megabytes* will increase automatically.

See page 118 to learn how to optimize the panel size.

If you use too small a value for *scratch_megabytes*, the DRC will crash with a message explaining the problem. However if you crash due a small scratch file during a preliminary DRC run on your design, you may want to try smaller panels before using a larger value for *scratch_megabytes*. This will conserve run time as well as memory.

Screen Display Options

None of these options affect how the DRC processes data or creates the output files. These options affect only the messages posted to the display to update the user on the progress of the DRC run.

[SHORTRUN] and [LONGCASE]

These optional keywords are used to choose the format of the console messages displayed while the DRC is executing. The SHORTRUN option optimizes the on-screen messages for short runs. It is intended for small designs that complete in a few minutes. This is the default behavior.

If your DRC run will take longer than a few minutes, you should add the LONGCASE keyword to the DRC command line. This optimizes the display when you will occasionally check on the progress of the run, rather than sit and wait for the DRC to finish.

See an example of an options file on page 336.

Since SHORTRUN is the default, use that keyword on the command line only when you want to override the LONGCASE keyword (e.g. when the LONGCASE keyword is in an options file.)

[DISPLAY_OPERATIONS=*min_refresh_seconds*]

This option and the following NO_FLASH_PANELS option are rarely used options to control the frequency of progress reports during the run.

By default, progress reports posted to the display during long runs are suppressed if it has been less than 2 seconds since the last progress report, even if the DRC has moved onto a new operation. Since the posting of these progress reports can take a significant amount of time for a long DRC run, you may want to increase this suppression interval for long runs. This option is especially useful if you do not intend to check the progress very often.

Example: **DISPLAY_OP=60**

Adding the option above to the DRC command line will suppress progress reports so that the display is updated no more often than once a minute.

If you prefer to have no suppression of the progress reports (the default behavior of previous versions of the DRC), add the following option to the DRC command line:

Example: **DISPLAY_OP=0**

This use of DISPLAY_OPERATIONS will force the DRC to operate as in previous versions. The progress display is more or less continuously updated. This option will also cause the NO_FLASH_PANELS limit (covered next) to be ignored.

[NO_FLASH_PANELS=*flash_limit*]

The DRC roughly estimates at the beginning of a run whether or not the run is likely to take a long time. The estimate is based on the number of panels and the number of rules. If this estimate is smaller than a default flash limit, then progress reports will flash on the screen more or less continuously. These updates to the display can increase the run time on the order of 15 minutes for a 7 hour DRC run.

(The actual progress update rate is controlled by the DISPLAY_OPERATIONS option described above. The default is to suppress updates to be no more often than every 2 seconds.)

The run time indicator estimate is calculated with the following equation:

See 118 to learn more about panels.

$$\text{Number_of_panels} * \text{number_of_rules_in_rule_set}$$

If the result of the calculation is less than the flash limit, then progress reports will be suppressed only by the interval limit defined with the DISPLAY_OPERATIONS option. If this estimate is larger than the flash limit, then most progress reports are suppressed entirely and the interval defined by the DISPLAY_OPERATIONS option is ignored entirely. The default flash limit is 100,000 for DOS versions, 10,000 for Windows versions.

You can set the flash limit directly by using the NO_FLASH_PANELS option on the DRC command line. If you prefer to suppress progress reports for even relatively short runs, decrease the flash limit to a smaller number. If you do not want any progress reports suppressed, then use the following command line option instead of a NO_FLASH_PANELS option:

Example: **DISPLAY_OPERATIONS=0**

[PAUSE=(ALWAYS | CRASH | NEVER)]

The console window will not close automatically when you type the command line in an open console window.

The following option is useful only when executing the DRC from a batch file. When you execute the DRC, without opening a console window explicitly, the program executes in a temporary console window. In this case, the console window may close automatically without giving the user time to see the final messages posted as the program terminates. Once the console window is closed, the console messages are gone forever. If the DRC crashes, it is very important to review these console messages to avoid repeating a wasted run.

Some users create a batch file with the DOS PAUSE command after the DRC command line to ensure that the console messages remain on the screen until

they can be viewed. With this version of the DRC, the extra PAUSE command can be omitted. If you prefer to hold the window open until the <Return> key is pressed, simply add a PAUSE option to the DRC command line.

PAUSE=ALWAYS	This will always pause the DRC after termination until the <Return> key is pressed.
PAUSE=CRASH	This option pauses the DRC after termination only when the program crashes. If the program terminates normally, no PAUSE is executed and no keystroke is required to close the program console.
PAUSE=NEVER	(This is the default if no PAUSE option is included on the DRC command line.) The DRC closes without an extra keystroke. If the console window will not close automatically when the DRC terminates, this is the best option.

Rules File Options

The following DRC command line options affect how the DRC rules file is processed and reported in the DRC log.

[LAYERS=(*layer_number1* [,*layer_number2* [...,*layer_numbern*]))]

You can write your rules file with variables for the ICED™ layer numbers rather than specifying the layer numbers explicitly. If your rules file uses this feature, you must supply the layer numbers at run time using this command line option. The layer number supplied for *layer_number1* will replace the %1 parameter wherever it is used in the rule set. *layer_number2* will replace %2, and so on. This allows you to create "canned" rule sets that you can run on any required layers at a later date.

Example: **DRC3-NT BLOAT10 XCHIP XCHIPOUT LAYERS=(3,10)**

Note that the parentheses are required.

If the rules file BLOAT10.RUL contains the following rules:

```
INPUT LAYER      %1 IN  
OUTPUT LAYER    %2 OUT  
OUT = BLOAT ( IN, .10 )
```

and the DRC command line is that shown above, the DRC will execute the rules as though they had been written as:

```
INPUT LAYER      3 IN ;  
OUTPUT LAYER    10 OUT  
OUT = BLOAT ( IN, .10 )
```

[DO=(*rule_spec1* [*rule_spec2* [...*rule_specn*]])]

A more complete explanation of rule sets is provided on page 152.

The DO keyword allows you to execute only specific rules from the rule set. You do not need to edit your rule set to execute only subsets of rules.

Each *rule_specn* parameter can be either the name of a rule set defined with the RULE_SET rule or a rule number. If you wish to execute all rules **except** for the specified rule number or set, place a dash in front of the specification.

Example: **DRC3-NT MYRULES XCHIP XCHIP SLOW DO=(FET_RULES)**

When this DRC command line is used with the RULE_SET example shown on page 296, the DRC will execute only the rules in the rule set FET_RULES, and any rules which generate the layers used in those rules.

You can combine rule set names and rule numbers after the DO keyword.

Running the DRC: Command Line Syntax

Example: **DRC3-NT MYRULES XCHIP XCHIP SLOW DO=(FET_RULES, 8)**

The rule numbers are listed in the rules compiler log file.

This command line will execute the rules in the FET_RULES rule set, rule number 8, and all rules required to generate the layers specified in those rules.

When rule set names are specified, you must surround rule specification with parentheses. When specifying only rule numbers the parentheses are optional.

Example: **DRC3-NT MYRULES XCHIP XCHIP SLOW DO=-8,-10**

This command line will execute all rules **except** for rule numbers 8 and 10. The layer generation rules required for only those rules will be skipped as well.

[SHOW_BORDER]

This command line option will add panel border calculations for each pass to the DRC log file. This can help you understand how your rule set affects the layer reach and panel border used by the DRC. If the panel border is large compared to the panel size, your DRC run may be very slow.

To learn about panels and borders, read "Panel Processing" beginning on page 118.

[BORDER=*border_dimension*]

The BORDER keyword is used to override the panel border.

WARNING: This option is intended solely for DRC experts. Border overrides can lead to incorrect DRC results.

You must thoroughly understand panels and borders before using this keyword. See page 124 for a complete explanation. **Setting too small a border can prevent the DRC from finding errors in your layout.** Setting too large a border can slow the DRC considerably.

Example: **DRC3-NT MYRULES XCHIP XCHIP SLOW BORDER=10**

This command line will override the default border calculated by the DRC or the border specified by any BORDER rule in your rule set. A panel border of 10 user units will be used by all passes. If this border is smaller than that calculated by the DRC, errors may not be found.

[NO_RUL]

Adding NO_RUL to your rule set has the same effect as adding this option to the command line.

The location of the source rules file and its time/date stamp are stored in the compiled rules file. If the time/date stamp of the source rules file is different than the information stored in the compiled rules file, the DRC will warn you, then ask you with a prompt if you want to proceed. This is to avoid a wasted run when you edit the rules file, then forget to compile it, before re-executing the DRC.

If the source rules file is not found in its original location, you will also receive an error message and a prompt. This is due to the fact that the DRC cannot tell if the source rules file is the same one that was used to create the compiled rules file.

If you want to suppress this warning message when the source rules file cannot be found, and avoid the prompt asking you if you want to proceed, add the NO_RUL keyword to your DRC command line.

This keyword will not prevent an error message and prompt when the source rules file is present but has a different time/date stamp than that stored in the compiled rules file.

Example: **DRC3-NT MYRULES XCHIP XCHIPOUT SLOW NORUL**

Note that the underscore is optional in the NO_RUL keyword. This is true of all keywords in the DRC command line.

[LIST_RULES]

Add this keyword to your DRC command line to add to the DRC log file a report of which rules were executed during each pass of the DRC. Using this keyword will also add to the DRC log most of the other information related to the rules file that is reported in the rules compiler log.

Example: **DRC3-NT MYRULES XCHIP XCHIPOUT QUICKPASS LIST**

Note that the LIST_RULES keyword can be abbreviated to "LIST".

[SHOW_SCALES]

The DRC calculates a default smoothing tolerance used by the BRIDGE rule. This tolerance allows the rule to recognize air bridge structures that are slightly less than perfect due to resolution grid rounding of the vertices. This default tolerance is usually .001 user units, but it can change if the design is very large. To see the actual default tolerance, add the SHOW_SCALES keyword to the DRC command line and search for "Smooth_tolerance" in the log file.

Design Area Options

Another method to limit the design area is to add the IN keyword to the DRC command in the layout editor.

The following keywords are used to allow you to verify only a portion of the layout data. You do not need to use all four options at the same time. Any combination of the following four options can be specified. If one of the options is not present, the default is to use the boundary of the design as the boundary of the area to check.

If your rule set specifies electrical connections for use by the MIN_SPACING /CONN or /~CONN keywords, these options may result in incorrect results since electrical connections may be made outside of the area used.

See page 100 for an example of the type of spacing errors the DRC may miss when QUICK-SPACING is used in combination with the design area options.

When these options are used, the DRC will automatically use the slower algorithm for spacing checks since vertices of shapes that violate a spacing check may be outside of the area checked. To override this slower algorithm, and risk missing spacing errors, you can also add the QUICK_SPACING option to the command line.

[LEFT=*left_x_coordinate*]

Add this option the command line to set the leftmost boundary of the area to be processed by the DRC. All shapes to the left of *left_x_coordinate* will be ignored.

See page 159 to learn how these options can lead to false errors being marked.

[RIGHT=*right_x_coordinate*]

Add this option the command line to set the rightmost boundary of the area to be processed by the DRC. All shapes to the right of *right_x_coordinate* will be ignored.

[TOP=*top_y_coordinate*]

This option sets the top boundary of the area to be processed by the DRC. All shapes above *top_y_coordinate* will be ignored.

[BOTTOM=*bottom_y_coordinate*]

Use this option to set the bottom boundary of the area to be processed by the DRC.

Example: **DRC3-NT MYRULES XCHIP XCHIP SLOW LEFT=410 RIGHT=1000 ...
... TOP=2000 BOTTOM=0**

This command line will force the DRC to use only the area of the layout file within a box with corner coordinates (410,0) and (1000,2000). The order of the boundary options is not important.

Cell Hierarchy Options

You should refer to page 134 to learn more about how hierarchical designs are processed by the DRC.

These options control how cells are flattened (i.e. ungrouped) hierarchically. The **FLATTEN**, **NO_FLATTEN**, **CFLATTEN**, and **NFLATTEN** options all control how cells are flattened **before** the DRC begins processing the data. When you use none of these parameters, the default DRC behavior is to flatten cells that are used only once, and those that have five or fewer components. This behavior tends to speed the DRC run by minimizing the repeat processing on small cells.

The **HIERARCHICAL** option affects only the cell hierarchy of the data created on output layers. When you do not add the **HIERARCHICAL** option to the command line, the output data will be created flat with no subcell structure.

[FLATTEN]

This option will force the DRC to ungroup all cells into one flat main cell before any layer processing is performed.

Example: **DRC3-NT MYRULES XCHIP XCHIPOUT SLOW FLATTEN**

See page 136 for more details on how the DRC handles dangerous operations.

This DRC command line will cause all cells in the **XCHIP.POK** file to be flattened hierarchically before the DRC processes the data. If the rule set calls for dangerous processing of cells or layers, this will be ignored since no cell hierarchy remains.

[NO_FLATTEN]

This option on the command line will prevent the DRC from flattening any cells. By default, all cells that have few shapes, and cells that are used only once, are ungrouped automatically. This command line option will prevent this ungrouping. The cell hierarchy of the entire design will be preserved.

[CFLATTEN=*component_count*]

Unless this option (or the NO_FLATTEN option) is used on the command line, the DRC will automatically ungroup all cells that have **five** or fewer shapes. When this option is used, the DRC will flatten cells that have *component_count* or fewer shapes.

Example: **DRC3-NT MYRULES XCHIP XCHIPOUT SLOW CFLATTEN=1**

When the option CFLATTEN=1 is used, all cells that have only a single component will be ungrouped by the DRC before layer processing begins. Other cells will remain hierarchically nested.

[NFLATTEN=*use_count*]

Unless this option (or the NO_FLATTEN option) is used on the command line, the DRC will automatically ungroup all cells that occur only **once**. When this option is used, the DRC will ungroup all occurrences of cells that are used *use_count* or fewer times.

Example: **DRC3-NT MYRULES XCHIP XCHIPOUT SLOW NFLATTEN=0**

When the option NFLATTEN=0 is used, no cells will be ungrouped because they are used infrequently. Cells used only once will remain hierarchically nested.

Example: **DRC3-NT MYRULES XCHIP XCHIPOUT SLOW NFLATTEN=4**

This command line will cause the DRC to ungroup all cells that are used 4 or fewer times.

[HIERARCHICAL="*suffix_string*"]

See page 372 to learn how to import hierarchical output data into the layout editor.

When this option is used on the command line, the DRC will create the shapes on output layers (except for error wires) in hierarchically nested cells. The cell structure will be modeled on the cell structure of the input data. However, cells that have been flattened with the options above will remain flattened in the output data. If you want the cell structure of the generated data to match your input data exactly, use the NO_FLATTEN command line option and the ALL_DANGER rule in addition to this option.

Safe handling of dangerous operations may put some shapes in a higher level cell than you would expect. See page 141.

Output from rules that produce error wires (e.g. MIN_SPACING, see the list on page 62.) will be created in a flattened main cell. Only polygon output shapes are nested in subcells.

The required *suffix_string* parameter will be added to the end of every cell name created in the output data. This prevents the newly created cells from modifying your original cells when you import the data with the *output_file_base_name.CMD* command file. You should add quotes around the *suffix_string* if it contains characters likely to confuse the DRC command line parser.

Example: **DRC3-NT MYRULES XCHIP XCHIPOUT HIERARCHICAL=_OUT**

The SLOW option is automatically invoked when HIERARCHICAL is used. Do not add the QUICK_PASS option to the command line.

This command line will cause the DRC to create shapes on output layers in nested cells. Each nested cell in the DRC database will have a cell created for it with the suffix "_OUT" added to the cell name. For example, if a nested cell in the input data has the name "SUBCELL", a cell will be created in the output command file with the name "SUBCELL_OUT". This cell will contain all shapes on output layers in the cell SUBCELL at the end of the DRC run.

You must execute the command file *output_file_base_name*.CMD to create the new cells. You should look at the layout carefully to make sure that it is what you expected.

Once you have determined that the layout generated by the DRC is exactly what you wanted, you can add the generated cells to your original data. The command file *output_file_base_name*.ADD can be executed to add each newly created cell to your original cells. You must execute this command file while editing a temporary cell. You cannot execute it from one of the design cells it will attempt to edit. (See page 374 for an explanation of the complete process.)

Example:

DRC3-NT MYRULES XCHIP XCHIPOUT HIERARCHICAL=""

To avoid a warning prompt when safe processing may create shapes higher up in the hierarchy than you would expect, use the NO_HIER-_WARNING rule.

This DRC command line is valid, but it can be hazardous. Since the *suffix_string* is "", the commands in XCHIPOUT.CMD will modify the original cells. No .ADD file is generated. If the results were not what you expected, you must exit the layout editor with the JOURNAL command to avoid saving the changes.

When the HIERARCHICAL option is not used, the output data will be created flat (i.e. no cell hierarchy) regardless of the cell structure of the input data.

Command File Options

To learn more about the command file created by the DRC see page 365.

The following command line options affect the *output_file_base_name*.CMD command file created by the DRC to create shapes in the ICED™ layout editor.

[WIRE_WIDTH=*error_wire_width*]

See the WIRE_WIDTH rule on page 315.

The WIRE_WIDTH rule sets the width used to create all error wires. If you need to override a value set with this rule, you can add this option to the command line. Alternately, if you do not have a WIRE_WIDTH rule in your rule set, then you can override the default behavior of creating error wires using

the default width of each error layer. All error wires created by the command file will be created using the width specified with this option.

When you set *error_wire_width* to 0, this removes the effect of any WIRE_WIDTH rule in the rule set and the DRC command file will create error wires using the default width of each error layer.

[START_CMD = "*st_cmdstring*"]

and

[END_CMD = "*end_cmdstring*"]

If you add the STARTCMD = "*st_cmdstring*" option to your DRC command line, the DRC will add *st_cmdstring* as the first line of the command file.

The ENDCMD = "*end_cmdstring*" option will cause the DRC to add *end_cmdstring* as the last line in the command file.

Both string parameters should be valid ICED™ layout editor command strings. You should surround the string parameters with quotes since blanks or '@' characters (used for the @*file_name* ICED™ command) would be misunderstood by the DRC command line parser.

Example: **DRC3-NT XRULES XCHIP XOUT START="EDIT CELL DRCOUT" ...
... END="EXIT;ADD CELL DRCOUT AT 0,0"**

This DRC command line will create a command file with the name XOUT.CMD. The DRC will add the following line to the top of the file:

EDIT CELL DRCOUT

When the EDIT CELL command is executed, the ICED™ editor will create a new cell with the name DRCOUT. All subsequent commands in the command file will create shapes in this new cell.

The command file will have the following two ICED™ commands added on the last line:

EXIT;ADD CELL DRCOUT AT 0,0

When these two commands are executed, the ICED™ editor will exit from the cell it has just created, then add that cell to the current cell at the coordinates 0,0.

See another example of the START and END options on page 370.

The total effect of adding these three commands to the command file is that all shapes created by the DRC command file will be created in a subcell of the current cell, rather than as components in the current cell. This is desirable since the command file will not then alter your original cells. Once you are finished looking at the DRC data, you easily can delete all of it by deleting the single cell DRCOUT. (If the DRCOUT.CEL file is saved to disk, you should delete it before a repeat DRC run. Otherwise, the next DRCOUT.CMD file will edit the existing DRCOUT.CEL leaving all of the errors found by the previous run in the cell.)

[OBSOLETE]

and

[MACROS=NONE]

These options are used to prevent the addition of macros and other special commands to the command file. These commands can cause problems to obsolete versions of ICED™. If you are importing command files successfully, you do not need them.

Panel Size Options

Use these options to override the panel size set in the rules file.

[PANEL_X=*panel_x_dimension*]

and

[PANEL_Y=*panel_y_dimension*]

Use these options to specify the panel size by x and y dimensions. These options have the same effect as PANELX and PANELY rules in the rules file.

[PANEL_A=*panel_area*]

See the next option to refine panel shape when area is specified.

You can use this option to specify maximum panel size by area rather than by x and y dimensions. Panel area is specified in one of three ways:

PANEL_A=*panel_area_in_square_user_units*

PANEL_KA=*panel_area_in_kilo_square_user_units*

PANEL_MA=*panel_area_in_mega_square_user_units*

[PANEL_X_BY_Y=*panel_ratio*]

and

[PANEL_Y_BY_X=*panel_ratio*]

If you use the previous option to specify panel area, you can optionally specify the ratio of the x and y dimensions with either of these options.

Batch Files

The recommended way to launch the DRC is to use a batch file. You use any text editor to create the command line with all of the desired command line options, and store it in a file with a .BAT extension. You then execute the batch file to execute the DRC command line. If you use many command line options this can save you a lot of typing and perhaps prevent wasted runs when you forget to add a critical command line parameter.

Another way to save DRC command line options in a file is to use command line indirection. See page 336.

The DOS SET command can be added to the batch file to set the DRC_PATH environment variable. This variable determines where the DRC will look for your rules file if it cannot find the rules file in the current directory.

You can use placeholders on the DRC command line to allow you to type some parameters on the batch file command line. DOS batch file placeholders must be a '%' followed by a number. For example, the lines below represent a simple batch file to launch the DRC program. Let us assume that the name of this file is DRCBAT.BAT

Example:

```
SET DRC_PATH=E:\ICED\DRCRULES  
DRC3-NT %1 %2 DRCOUT SLOW PAUSE=ALWAYS %3 %4 USE=8000
```

The first two placeholders in the command line mean that you must specify the name of the rules file and the layout data file on the command line when you execute the batch file. The additional placeholders allow you to add other parameters on the batch command line that will be added to the DRC command line after the PAUSE option.

Example:

```
DRCBAT MYRUL MYPOK QUICKPASS
```

When you type this command line at the DOS prompt, the DRC will execute with the following command line:

```
DRC3-NT MYRUL MYPOK DRCOUT SLOW PAUSE=ALWAYS...  
... QUICKPASS USE=8000
```

The QUICKPASS option will override the SLOW option.

Running the DRC: Command Line Syntax

If your operating system is pre-parsing the command line and replacing the '=' with a blank, use '#' instead. See page 333.

The PAUSE option in the example above is used primarily when you execute this batch file without opening a console window explicitly. If you execute the batch file with a method like a desktop shortcut, a temporary console window is opened to display the console messages. However, once the DRC completes, the temporary console window will close immediately unless the PAUSE option is added to the command line. The PAUSE option will prevent the window opened for the console from closing until you have a chance to view the console messages. Just press <Enter> to close the console window.

Another DOS command you may want to add to your batch file is a CD (or CHDIR) command to change the current directory before the DRC begins. Any DOS commands you desire can be added to the batch file.

DRC Output Files

The DRC can generate several output files. The file name extensions vary from file to file, but the base part of the file name for most of the files will be the string provided in the *output_file_base_name* parameter (the third parameter) on the DRC command line.

If you do not provide an *output_path* with the *output_file_base_name* parameter, all files will be created in the current directory.

If a previous DRC run has created output files in the same directory, most of those files will be renamed with a '1' replacing the final character in the file extension before the new files overwrite them. This provides a backup of the results of your last DRC run with no effort on your part.

	File extension	Contents and use of file
DRC log file	.DLO (previous run backup .DL1)	General information about DRC run All error and warning messages Statistics on run time
DRC command file	.CMD (previous run backup .CM1)	Command file used to create shapes on output and error layers in the ICED™ layout editor
Hierarchical cell command file	.ADD (previous run backup .AD1)	Command file used to add hierarchical output to original cells
Subcell error command files	.ERR (file names are based on cell names, if possible) (previous run backups .ER1)	One command file is created for each input cell that contains errors found in the subcell (including bad polygons and dangerous processing warnings).

Figure 211: DRC output files.

Running the DRC: Output Files

The DRC log file will record the file names used for each of these files.

Two or more temporary files are created by the DRC. A file with the name DRC3CMD.\$\$\$ is created in the current directory, then erased as the DRC begins. One or more scratch files with the name \$D3VIRT.000 are created depending on the use of the SCRATCH_DIR command line option. These files will be deleted automatically by the DRC at the end of a successful run.

Due to the creation of these temporary files, it is unwise to launch multiple runs of the DRC simultaneously from the same directory. This could be a problem on networks or multitasking operating systems. Simultaneous runs of the DRC should have no problems as long as they are started from different directories.

DRC Log File

This file is where the DRC will store all error and warning messages. All of the information printed on your screen as the DRC is executing (except for the progress indicators) will be recorded in this file at the same time. In addition, many detailed error messages and counts of shapes created on all output and error layers will be reported.

This file **will not** contain the coordinates of all errors found in the layout. The layout error data is represented in the DRC command file (covered on page 365). You can enable detailed logging for a few of the verification rules. This will add a detailed message about each error found (including coordinates). See page 50 for a complete description of detailed logging.

If you use the following DRC command:

DRC3-NT MYRULE MYCELL DRCOUT

The name of the DRC log file will be DRCOUT.DLO.

The log file begins with a header that includes the version number of the program. The command line used to execute the program is then reported. This information is followed by the names of the input and output files.

To include a complete listing of the rules file at this point in the DRC log, add the `LIST_RULES` keyword on the DRC command line. The information added will include a complete list of input, output, and scratch layers and the rules executed in each pass.

The amount of memory available to the DRC program is listed next along with the date and time of the run.

If the `SHOW_SCALES` option is included in the DRC command line (see page 350), then the log file at this point lists several scale factors used by the program.

Bad polygons
are defined on
page 74.

If bad polygons (shapes which may cause problems for mask processing software) are included in your layout, they will be mentioned next in the log. Each cell that contains bad polygons will have a command file created for it that has the bad polygons copied to an error layer. These files are described in more detail on page 374. (The total number of bad polygons is also listed in the summary information along with the final error count near the end of the log file.) The listing and copying of bad polygons on input layers that are not used in your rule set can be turned off by the `NO_CHECK_INPUT` rule. See page 276.

Some details on how the layout was ungrouped (i.e. flattened) are printed next. If your layout contains cells that are not ungrouped because they are used in an `INCELL` rule, or because of cell flattening options or defaults (see pages 144 and 352) this will be reported.

The cut resolution in effect is listed next. This is set by the `CUT_RESOLUTION` rule. (See page 79 for a description of the various resolution grids.)

See page 118 to
learn more
about panels.

The design area coordinates are listed here, as well as the panel size used by the DRC.

The number of operations and passes required by the rules is listed next. (See page 128 for an explanation of passes.) The DRC algorithm chosen is listed here, either "Slow method" or "Quick pass" will be listed. (See page 129 for details.)

The string "Flat output" or details on the hierarchical output options will be reported. (See page 146.)

Next, the log lists the spacing method used. This method is determined automatically by the rule set. However, the QUICK_SPACING DRC command line option can overrule the slower spacing method at the risk of missing errors in certain rare cases. (See page 100.) When the QUICK_SPACING option has overridden the slower method, the log file will print a "**** WARNING" message about this.

The panel border used for each pass is reported next. Add the SHOW_BORDER keyword to the DRC command line to see how the panel borders were calculated.

The cells checked and their coordinates will all be listed. (Cells flattened by the pre-processor are listed above.)

See page 313
for details on
acute angle
warnings.

If the NO_WARN_ACUTE rule is not present in your rule set (see page 280), all acute angles on polygon output layers will be listed here.

At this point in the log file, the DRC will add the following line at the conclusion of a successful run:

Done.

If bad polygons were reported earlier in the log file, the total number will be summarized at this point. A summary of acute angles is listed next.

If no shapes were created by the DRC on error layers (i.e. no errors were found by verification rules), the DRC will post the following message:

*****No errors found.**

If errors were found, the total number of error shapes generated by the DRC is provided and looks similar to:

*****Error count=76**

The DRC will often create a pair of error shapes for a single violation. Overlapping error shapes on the same layer may be merged. For these reasons, the number of error shapes generated is not the same as the number of rule violations found.

The log file will continue with a summary of shapes created on all output and error layers.

The log ends with statistics on the scratch file and processing times. If a large scratch file was required, you may be able to decrease your run time significantly by reducing the panel size. One indicator for DRC efficiency is the percentage of time the DRC spent on "Disc swaps". If more than 50% of the run time is spent swapping information into and out of the scratch file, you should read the information beginning on page 118 to learn how to optimize panel size.

The time spent processing each rule is presented here as well. The times are listed by operation number. To relate the operation number to the rule, look in the rules compiler log file or add the LIST_RULES option to the DRC command line.

DRC Command File

Read "Generating Output Layers" on page 70 for important information on using the DRC to create new mask layers for import into your design.

This file will contain ICED™ layout editor commands that create shapes to indicate the errors the DRC has found and/or to import the results of layer processing. The DRC will create this file with the name *output_file_base_name*.CMD (where *output_file_base_name* is the third parameter on the DRC command line).

If you use the following DRC command line:

DRC3-NT MYRULE MYCELL DRCOUT

then the name of the DRC command file will be DRCOUT.CMD.

Running the DRC: Output Files

The LOCAL and GLOBAL commands in the DRC generated command file create macros (similar to variables). Lean more in the ICED Command File Programmer's Reference Manual.

The SHOW command in the layout editor can be used to report the tag number of a selected shape.

Using a WIRE_WIDTH rule makes changing the layers in the cell unnecessary. However, you should still set the default wire type with the USE command.

Every shape on layers defined with the OUTPUT LAYER rule will have an ADD command in the file. These shapes are created from the layer data at the end of the DRC run.

The ADD commands in this command file (and all command files generated by the DRC) are written in ASCII. You can browse this file to see the coordinates of each error shape if you desire.

The command file will have comments in it to aid you in determining which rule of your rule file generated the shape. The comment before a block of ADD commands will contain the string "TAG=rule_number". The *rule_number* parameter is equal to the number of the rule that created the shapes. The rule numbers are listed near the bottom of the rule compiler log.

Before Executing the Command File

You may want to set the layer properties of the layers to which you will be adding shapes before executing the command file. You set the name, color, fill pattern, and default width for each layer number with **LAYER** commands in the layout editor. You set the default wire type for all layers with the **USE** command. It is particularly important to set the **default width** and **wire type** of error wire layers before executing the command file. It is tricky to change these properties after the shapes are created.

Always set the default wire type of error wires to type 0. Type 2 wires can be very confusing since the wire will extend past the error.

Set the default width of error wire layers to be small compared to the minimum width of the design layer they will be marking. This will allow the error wires to distinctly mark only the edges of shapes that are in error. If the default width of an error wire layer is large compared to the width of the shapes, the error wires will overlap and be difficult to see.

(If you prefer to avoid customizing the wire widths of all error layers in the cell before you import the shapes in the command file, add a WIRE_WIDTH rule to the rule set. When this rule is used, all error wires will be created with the same specified width.)

To see the current properties of a layer in the layout editor, type the LAYER command with the layer number as the only parameter.

```
USE WIRETYPE=0  
LAYER 50 NAME=M1_ERROR WIDTH=.2 COLOR=RED
```

Executing these two commands in the ICED™ layout editor **before** executing the DRC command file will set the default properties of the error wires about to be created on layer 50. The default width of .2 user units is a good dimension if the M1 layer in your design has a minimum width of 1 user unit or more.

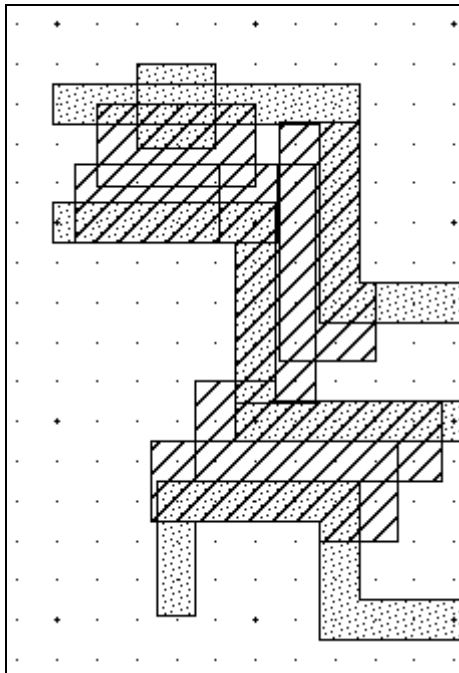


Figure 212: Confusing error wires of type=2 and width=2.

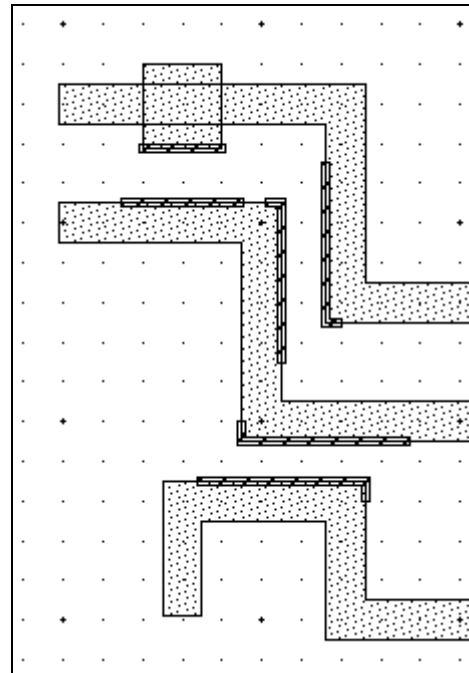


Figure 213: More distinct error wires of type=0 and width=0.2.

Running the DRC: Output Files

You can add the @ERRWIRE.CMD command to the DRC command file with the START_CMD option on the DRC command line. (See page 355 and the examples below.)

You can create a command file to set the properties of error layers. Simply create an ASCII file with the USE and LAYER commands you desire. If you create this file with the name ERRWIRE.CMD in the current directory, you execute it in your cell with the layout editor command:

@ERRWIRE.CMD

Executing the Command File in the Layout Editor

Unless you have used the HIERARCHICAL keyword on the DRC command line, there will be no cell structure in the generated DRC shapes.

The same @file_name command syntax as that used above is used to execute the DRC command file. To add the shapes in the file DRCOUT.CMD to the cell you are currently editing, type the layout editor command:

@DRCOUT.CMD

However, **executing this file in your design cell will modify your design cell.**

See page 372 learn about important differences in the command file when the HIERARCHICAL keyword is used on the DRC command line.

If your rules output only layer numbers that are not used in your design, it is relatively easy to delete the DRC generated shapes. You can delete all shapes on a given layer number *n* with the layout editor commands:

**UNSELECT ALL
SELECT LAYER *n* ALL
DELETE SEL**

You can create a command file with a name similar to DELDRC.CMD with all of the commands necessary to remove all of your DRC layers. Simply execute this command file when you are done looking at the DRC shapes, before you save the cell files. (You may want to add the command XSELECT=OFF to the top of this command file. See the layout editor reference manual.)

Isolating the DRC Shapes from the Original Layout Data

If you prefer to isolate the shapes generated by the DRC from your design cells, there are methods that allow you to view the DRC generated shapes without affecting your original design cells.

One simple way is to create a new cell with the ICED™ layout editor, then execute the @DRCOUT.CMD command in this new cell. You can then add the new cell to your top-level cell, or add both the new cell and your top-level design cell to a different new cell. Usually, the best method is to add your top-level design cell to the new cell containing the DRC generated shapes. This will allow you to turn the display of your design shapes on and off as you view the DRC shapes.

The VIEW OFF and LOG SCREEN OFF commands in the DRC generated command file are present to speed up execution. Learn more in the layout editor reference manual.

Launch the ICED™ layout editor to create a new cell with a name like NEWCELL

Add the DRC shapes with the ICED™ command:

@DRCOUT

(The file extension of .CMD is added automatically to the command file name by the @file_name command when no file extension is provided.)

You can then add your design cell to the new cell with a command similar to the following ICED™ command:

ADD CELL MYCELL AT 0,0

Running the DRC: Output Files

You can use the START_CMD and END_CMD options on the DRC command line to make this type of processing automatic. These options add layout editor commands to the DRC command file.

```
!DRCSTART.CMD  
!Initializes defaults for error wire layers  
USE WIRETYPE=0  
LAYER 50 NAME=M1_ERROR WIDTH=.2 COLOR=RED PAT=7  
LAYER 51 NAME=M2_ERROR WIDTH=.2 COLOR=BLUE PAT=12  
LAYER 52 NAME=P1_ERROR WIDTH=.1 COLOR=GREEN PAT=14
```

Figure 214: Command file DRCSTART.CMD

```
!DRCEND.CMD  
!Add top-level design cell to temporary cell with DRC output  
ADD CELL MYCELL AT 0,0
```

Figure 215: Command file DRCEND.CMD

You can create the two command files shown above with any ASCII text editor. Then execute these command files automatically whenever you execute the DRC generated command file when you add the following options to your DRC command line:

START="@DRCSTART" END="@DRCEND"

See a another example of using START and END on page 356.

The command @DRCSTART will be added to the DRC command file so that the commands in DRCSTART.CMD are executed before the ADD commands generated by the DRC program. The commands in DRCEND.CMD will be executed after the last ADD command generated by the DRC. You can modify the DRCSTART.CMD and DRCEND.CMD files as you develop your own methods for viewing the DRC output.

Making the DRC Shapes More Visible

Sometimes the shapes the DRC creates are difficult to see since they are copies of shapes in your design cell which are right on top the original shapes. There are several ways to make the shapes easier to see.

One way is to temporarily turn off the display of your design cell with the command:

BLANK CELL LAYERS 0:255

You can turn display of your design cell back on with the command:

UNBLANK ALL

You can use color to highlight shapes on the DRC layers with the ICED™ LAYER command. A good color to use for the DRC layers is the HI color. This color will always have priority on your screen so that shapes drawn with that color are not hidden behind other colors. To assign the color HI to a layer generated by the DRC command file, use the command:

LAYER *n* COLOR=HI

where *n* is replaced with the layer number in the DRC command file.

You can then make this color strobe on and off with the command:

BLINK

Another way to locate shapes on one of the DRC layers is to select only the shapes on that layer with the commands:

**UNSELECT ALL
SELECT LAYER *n* ALL**

You can then resize the view screen to see all selected shapes with the command:

VIEW SELECT

Determining Which Rule Generated a Shape

If you see an error shape on your screen, and you don't know which DRC rule generated the error, you can use the SHOW command in the layout editor to report the TAG number. Select the shape, then type:

SHOW *

Rule numbers are reported in the rules compiler log file.

The SHOW command will report the TAG number of the selected shape. Remember that the TAG number is the number of the rule that generated the shape.

Fixing the Errors

Once you have located a DRC generated shape which points out an error in your design cell, you can edit the cell which contains the original design shape, without exiting the editor, by using any of the ICED™ edit commands: EDIT, PEDIT, or TEDIT.

The easiest edit command to use is usually the PEDIT NEAR command. After typing this command, place your cursor on an edge of the design shape you need to modify to correct the problem, then click the left mouse button. You are now editing the cell that contains the design shape, even if it is nested several levels down. The shapes in other cells will remain on the screen but you will not be able to edit them. If you prefer that shapes in other cells are not displayed while you edit the cell with the problem, use the TEDIT NEAR command instead of PEDIT NEAR.

Hierarchical Output

Unless you have used the HIERARCHICAL keyword on the DRC command line, the DRC command file will create all shapes in whatever cell you execute it from. There will be no cell structure in the generated data.

When you do use `HIERARCHICAL=suffix_string` on the command line, the DRC command file will create all shapes in cells which match the cell structure of your original data. The cell names will all have the *suffix_string* added to the end of the original cell names.

Example:

DRC3-NT MYRULES XCHIP XCHIPOUT HIERARCHICAL=_OUT

Refer to page 146 for an important overview of how hierarchical output is handled by the DRC.

When this DRC command line is used, the command file XCHIPOUT.CMD will generate output layer shapes (except for error wires) in separate cells by using the layout editor's EDIT command. If a cell with the name SUBCELL is included in the original data, a cell with the name SUBCELL_OUT will be created in XCHIPOUT.CMD with commands similar to those shown below:

```
EDIT CELL SUBCELL_OUT
ADD BOX LAYER=20 AT (0,0.6) (1.8,5.9)
ADD BOX LAYER=20 AT (4.2,0) (6.9,5.6)
ADD BOX LAYER=20 AT (9.8,0.5) (12.4,5.6)
EXIT
```

When these lines from the command file are executed in the layout editor, a new cell with the name SUBCELL_OUT is created.

Shapes on error wire layers will be created in the highest-level cell, the cell with the name based on the name of your top-level cell. If the name of your top-level cell is MAIN, the error wires will all be created in a new cell with the name MAIN_OUT. Errors that are indicated with polygons rather than wires will be created in the cell corresponding to the cell with the error. (See the chart on page 62 to see which rules create wires.)

Executing the XCHIPOUT.CMD file will create the new cells, however, it will not add them to your original cells. In fact, after you have executed the command file, you will see no new data in the layout editor. The cells have been created, but none of them are added to the cell you are currently editing in the layout editor. To look at one of the new cells, you will have to edit it with the EDIT command in the layout editor, or EXIT and then re-launch the editor to edit the new cell.

If you use the null string "" as the *suffix_string* in the HIERARCHICAL specification, the DRC command file **will** edit the original cells. In this case,

executing the XCHIP.OUTPUT.CMD file will modify your design cells. **This can be very dangerous if you do not know exactly what you are doing!** Make sure you have your design backed up before attempting to use this feature. You will be modifying your design cells before verifying that the DRC has generated exactly the data you intended.

When the *suffix_string* is "", you will have to execute the command file while editing a temporary cell. An EDIT command in the command file will fail if you attempt to execute it while already editing the cell.

To add the new cells to your original design cells, you execute another file created by the DRC, the hierarchical cell command file with the name *output_file_base_name.ADD*. We will now describe how to use this file.

Hierarchical Cell Command File

Read an overview of hierarchical output on page 146.

This file is used to add cells created by the `HIERARCHICAL=suffix_string` command line option. It is created only when you have used the `HIERARCHICAL` option. It is a command file of ICED™ layout editor commands to add each cell created by the DRC to your original cells.

We strongly recommend that you read example in the Advanced Tutorial on page 433 before attempting this in a real design.

The name of this file will be *output_file_base_name.ADD*. You must execute it in the ICED™ layout editor while editing a new temporary cell. This is due to the fact that the command file uses the EDIT command to open each of your design cells. The command file will fail if it attempts to EDIT a cell that is already open in the layout editor.

Your original cells will be modified by this operation. Be sure that the data in the new cells is what you expect before executing the .ADD file.

Example:

DRC3-NT MYRULES XCHIP DRCOUT HIERARCHICAL="NEW"

If you have used this DRC command line, and your design contains a cell with the name "SUBCELL", the DRC command file DRCOUT.CMD will create a cell with the name SUBCELLNEW. Once you execute the .CMD file, the cell is

created, however this new cell has not yet been added to your design. You should look at the contents of this new cell (and any other new cells) before executing the DRCOUT.ADD file. The .ADD file will contain layout editor commands that add the cell SUBCELLNEW to cell SUBCELL.

Use the TEMPLATE command in the layout editor to export environment settings from a design cell. However, if your startup command file is set up correctly, all new cells are created with an appropriate environment automatically.

You must open a temporary cell with the ICED™ layout editor to execute this file. The temporary cell should contain appropriate environment settings since the environment settings (e.g. layer colors and patterns) in effect will replace the environment settings in all cells modified by the command file. In the editor, type the following string:

@DRCOUT.ADD

Always execute the .CMD file before executing the .ADD file.

The cells created by the .CMD command file and then added by the hierarchical command file can be time consuming to remove again once the command files are executed and the cell files saved to disk. If you realize that the added cells are incorrect, terminate the ICED™ layout editor with the JOURNAL command to avoid saving cell files.

If you have added cells from a previous DRC run and need to create a new set that is slightly different, be sure to not only delete each DRC generated cell from each original design cell, but also delete the previous run's cell **files** before executing the new command file. Otherwise, the new command file will add new shapes to the previous cell files rather than creating new cell files.

Subcell Error Command Files

The log file will tell you if shapes have been created in these files.

There are several types of errors that the DRC can mark at the subcell level. These types of errors are found by the DRC while the cells remain hierarchically nested. Since the errors should be fixed in the subcells, the DRC places the error shapes in command files that you should execute within the subcells.

Running the DRC: Output Files

The following types of errors will produce shapes in these subcell error command files:

Bad polygons (See page 74.)

Dangerous processing errors (See page 140.)

When these types of errors are found, you can use these files to add copies of these problem shapes to your cells on a new layer in a manner similar to the other DRC command file described on page 365.

One command file is created for each cell that contains these types of errors. The name of the command file will be *cell_name*.ERR.

The coordinates used in the ADD commands contained in the subcell error command files will be in the coordinate system of cell *cell_name*, not in the coordinates of the top-level cell. **Run each command file while you are editing cell *cell_name*, not the top-level cell.**

The layer number for bad polygons can be overridden with the BAD_POLY rule.

For example, you run the DRC on your highest level cell, MAINCELL. The cell NANDCELL is used 100 times in MAINCELL. NANDCELL contains a single self-intersecting polygon. You get only a single warning message about the bad polygon, not 100 messages. The DRC will create a command file with the name NANDCELL.ERR. This command file will contain a single ADD command that creates a copy of the bad polygon on the error layer 99.

To see exactly which shape is causing the error message, you can edit the cell NANDCELL with the ICED™ layout editor, then run the command file with the ICED™ command:

@NANDCELL.ERR

See page 365 to learn other ways to use DRC generated command files.

This will add the shape on layer 99 to the cell. Now you can see the exact problem shape with the commands:

**SELECT LAYER 99 ALL
VIEW SELECT**

This will add select marks to the shape(s) just created and resize the view window so that all shapes on layer 99 are displayed. Once you see the shape that is causing the problem, delete the shape(s) on layer 99 and fix the original shape.

Running the DRC: Output Files

Advanced Tutorial

Advanced Tutorial

This tutorial covers most of the typical activities involved in using the DRC to verify common technology design rules. We will cover the steps to compile the rules and create the layout file for the DRC rather briefly. To see a tutorial that covers these steps more completely, see the Quick Tutorial beginning on page 12.

The layout files for this tutorial should be included with the installation. The rules files will be created from scratch.

Subject	Page
Generating temporary layers for verification	383
Simple spacing check	383
Importing DRC results	387
Directional spacing checks	391
Errors from touching shapes	397
Electrical connections	402
Masking an input layer	403
Well connection verification	411
Export of mask layers	418
Finding/Fixing acute angles	423
Finding/Fixing bad polygons	426
Hierarchical Output	429
Dangerous operations	430
Speeding DRC runs with rule subsets	440
Pad size verification using MIN_WIDTH rule	441
Speeding DRC runs by optimizing panel size	445

Simple Spacing Check

Subjects covered below
Basic steps to generate input files
Generating temporary layers for verification
Simple spacing check
Setting default width for error wires
Basic steps for importing DRC results into layout editor
Fixing errors in subcells

First, let us briefly review the steps to prepare the input files for the DRC. We will need to create a text file with the rules for DRC and compile this file. We also need to create the binary layout data file from the cell files using the ICED™ layout editor. We will perform this step first.

Create a new directory for this tutorial. The name is not important, but we will use the name Q:\ICED\DRCTUTR to refer to this directory throughout the tutorial. You need to copy the tutorial cell files into this new directory.

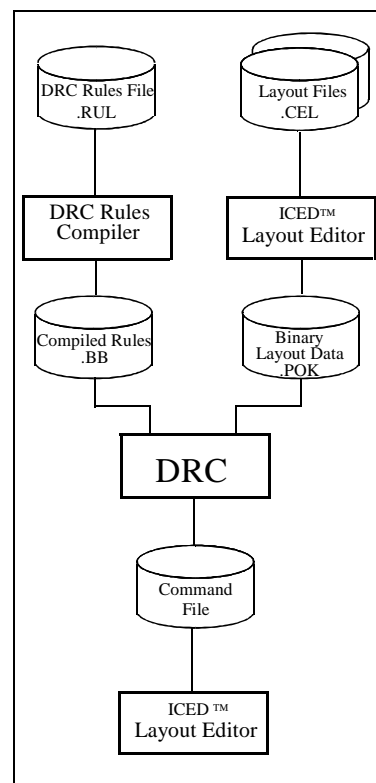


Figure 216: DRC data flow.

Open an appropriate console window by clicking the ICED icon on your desktop. Type at the console prompt:

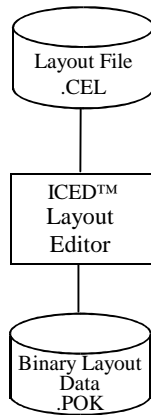
```

MD DRCTUTR
CD DRCTUTR
COPY Q:\ICED\22SAMPLES\DRC\ADV*.CEL

```

²² Q:\ICED represents the drive letter and path where you have installed the DRC.

Preparing the Binary Layout Data File



Launch the ICED™ layout editor to edit the file ADVTUTR.CEL. If you use the Windows version, launch the layout editor with the following command in the console window:

ICWIN ADVTUTR

The shapes in this cell should look similar to Figure 4.

Once in the editor, we create the binary layout data for the DRC using the DRC command. Type the following on the command prompt line:

DRC

This will export the entire layout contained in the cell to the ADVTUTR.POK file. Then use the **QUIT** command to terminate the editor. (If you using a multitasking operating system, you can leave the editor open and return to it later.)

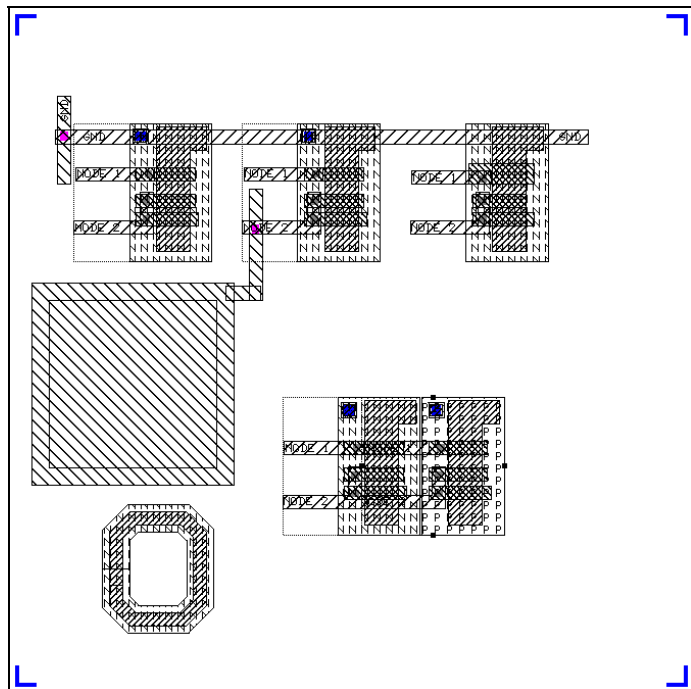
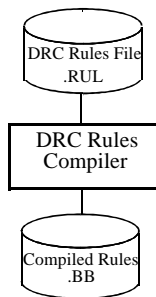


Figure 217: ADVTUTR.CEL

Creating the Rules File for a Simple Spacing Check on a Generated Layer



Let us assume that you are working with a CMOS technology that forms transistor gates where a layer with the name “poly” crosses a layer with the name “active”. You need to verify that every gate is at least 2 microns away from any contact hole indicated by the cpoly layer.

To create the DRC rule set to check this spacing rule, we first need to create an ASCII text file to contain the rules. Create the file with any method, but be sure to store it in the ADVTUTR directory. If you use DOS, you can use the following command:

EDIT ADVTUTR.RUL

Type the lines shown in Figure 219. Note that all layer names are in lowercase and all DRC keywords are in uppercase. This is to help you identify which text represents layer names. If you copy these lines for another use later on, you can change the layer names to any convenient strings, but the DRC rule keywords should remain as typed. The unbolded text represents comments.

```

ALL_SAFE

INPUT LAYER 4 active !Diffusion
INPUT LAYER 8 poly !Poly
INPUT LAYER 9 cpoly !Contact from Metal to Poly

OUTPUT LAYER 0 gate !Transistor gate

OUTPUT LAYER 101 gate_cpoly_sp_err
WIRE_WIDTH=0.3

gate = active AND poly

gate_cpoly_sp_err = MIN_SPACING (gate, cpoly, 2)
  
```

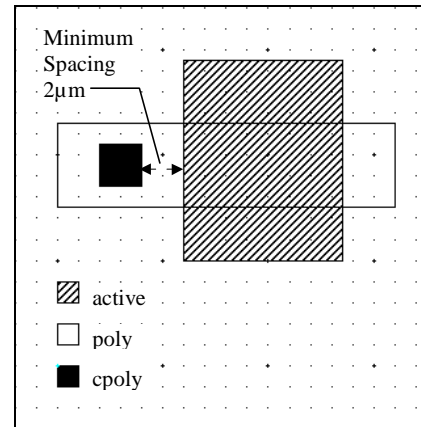


Figure 218: Poly contact to gate spacing rule.

Figure 219: Initial contents of the ADVTUTR.RUL file.

Advanced Tutorial: Simple Spacing Check

Dangerous processing options are explored on page 430.

The first rule in a rule set is usually the choice between safe and dangerous processing. The “ALL SAFE” rule should be the first rule in any rule set unless you need dangerous processing that creates shapes hierarchically.

The layer definitions come next. All layers used in the rule set must be defined, including the layers used to mark errors. The WIRE_WIDTH rule sets the size for all error wires created by the DRC.

The AND rule near the bottom of Figure 219 creates shapes on the scratch layer “gate” which represents the overlap of the active and poly layers. This scratch layer must also be defined before it is used. This scratch layer is defined with the rule “OUTPUT LAYER 0 gate”. It is a good idea to create scratch layers as output layers with a layer number of 0. This allows you change the layer to an output layer in a later run by changing only the layer number in the OUTPUT LAYER rule. You may need to look at the shapes on a scratch layer to diagnose problems with the rule set. Any output layer with the layer number 0 will **not** create shapes in the output.

The final rule in our rule set is the “gate_cpoly_sp_err = MIN_SPACING (gate, cpoly, 2)” rule. This is the rule that tests the spacing between the gate layer and the cpoly contact layer. If sides of a shape on the cpoly layer are closer than 2 microns away from sides of a shape on the gate layer, then error marks will be created on the gate_cpoly_sp_err layer. This layer is defined above with the layer number 101. It is best to use layer numbers not easily confused with design layer numbers for error or temporary layers. This allows you to easily delete all imported shapes on these layers from your layout.

Save the rules file. If you are using DOS, exit the editor. (If you are using a multitasking operating system, you can keep this window open to continue to make edits to the file as we proceed through the tutorial.)

This rules file must be compiled by the DRC rules compiler. The compiler command line can be typed in the console window opened by the ICED desktop icon, or stored in a batch file. If you store the command line in a batch file, you can execute the compiler by executing the batch file. You can execute the batch file without explicitly opening a separate console window.

To create a batch file, create another text file containing the command line below. Add the PAUSE=ALWAYS option to the end of compiler command line to keep the temporary console window open long enough to see any compiler messages. The console window will remain open until a key is pressed. Save the file with a name similar to DRCRULES.BAT.

If you do not use a batch file for the compiler command line, type the command line at the prompt in the console window opened by the ICED icon on your desktop.

Type the following command line:

D3RUL-NT²³ ADVTUTR

D3RUL-NT.EXE is the name of the compiler program and the rules file is ADVTUTR.RUL. This program will create the compiled rules in a file named ADVTUTR.BB. We will use this file later when we run the DRC.

The console messages will be very brief. The version of the compiler, along with a copyright notice is followed by a report of how much memory is available to the compiler. When the compiler finds no errors, the next line is:

Done.

This indicates a successful compile.

If you type a mistake in a rules file, you will get an error message from the compiler. For example, if you mistype the layer name in the AND rule and use “ppoly” instead of “poly”, you will see an error message similar to the following in the console messages and in the rules compiler log file ADVTUTR.RLO:

```
Error in file Q:\ICED\ADVTUTR\ADVTUTR.RUL, line 11, column 19:
gate = active AND <ppoly>
Layer name expected.
```

The item enclosed in <> is the item in error. Fix any typing errors in your rule set and recompile until the rules file successfully compiles.

²³ The executable file for released versions for Windows is D3RUL-NT.EXE. The executable file for Beta Windows versions is named D3RU-NTX.EXE.

Executing the DRC

Type the DRC command line in the console window opened by the ICED desktop icon or create a batch file. Add the PAUSE keyword to the end of the DRC command line if you use a batch file.

If you see an error message from the DRC about “not enough free memory to run”, try adding an appropriate HOG option to the command line. See page 339.

Now we are ready to run the DRC. The DRC command line is:

DRC3-NT²⁴ ADVTUTR ADVTUTR DRCOUT SLOW

The program DRC3-NT.EXE will execute using the ADVTUTR.BB compiled rules file and the ADVTUTR.POK binary layout data file. All output file names will begin with “DRCOUT”. The SLOW keyword is required to make the choice between quicker algorithms that may miss errors in some rare cases and the slower but more accurate algorithms.

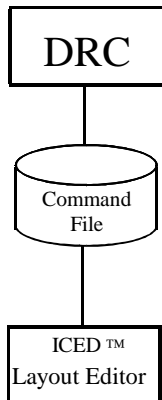
The end of the console messages should look similar to:

```
Done .  
  
100% of chip done.  
***No input skipped.  
***No bad ICED polygons.  
***Error count=8 (raw=20)  
***Total output non-error output count=0  
    0 total figures output to non-error layers.  
    8 total figures output to error layers.
```

This indicates that the DRC has generated 8 error marks. Most spacing errors generate a pair of error marks. When error marks on the same layer overlap, they are merged into single shapes. A single error in a subcell creates error marks in the flattened main cell for each instance of the subcell. After looking at the error marks, we will see that these 8 error marks represent 2 errors in the layout. We will now cover how to locate these errors from the command file generated by the DRC.

²⁴ The executable file name for released versions for Windows is DRC3-NT.EXE. The executable file name for beta Windows versions is DRC3-NTX.EXE.

Looking at the Output



To view the errors found by the DRC, we will use the ICED™ layout editor. Launch the layout editor again to edit the ADVTUTR cell. You can use the same command we mentioned earlier:

ICWIN ADVTUTR

To execute a command file generated by the DRC, you use the *@file_name* command in the layout editor. To execute the command file created by our tutorial, type the following command in the editor:

@DRCOUT

This will execute the DRCOUT.CMD command file generated by the DRC. (The name of this file was determined by the third parameter on the DRC command line.)

The 8 error wires are now added to the cell. Even with a simple layout, it can be difficult to spot the error marks. To select the error wires just added to the cell, use the following editor command:

SELECT LAYER 101 ALL

Hopefully you can now make out where some of the error wires are present. Let us zoom in on one of these spots by typing the following command (or selecting it from the menu):

VIEW BOX

Now select view box corners near the points indicated in Figure 220.

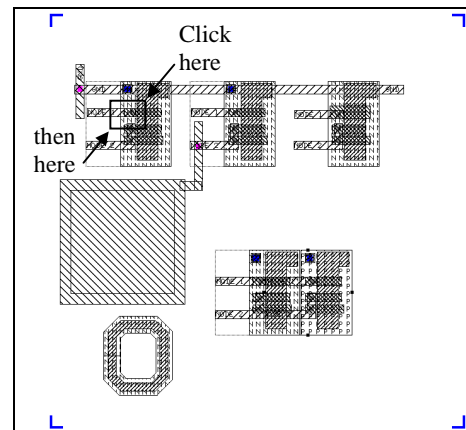


Figure 220: Defining zoom view.

Advanced Tutorial: Simple Spacing Check

Use the **BLANK** command in the layout editor to hide the display of certain layers.

If it is still difficult to see the error marks, you can blank layers Metall and Nwell as we have in Figure 221.

Note that the error wires wrap around the edges of the shapes in error. This is because the indicated parts of those sides also violate the 2 μm spacing rule. The error wire wraps around the edges of the shape on the gate layer even though this shape is not present in either the cell or the data created by the DRC.

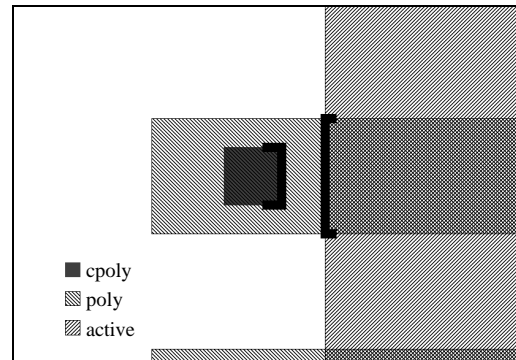


Figure 221: Error wires.

If you did want the gate shapes created in the cell, you could change the gate layer number in the rule set to a non-zero number and re-execute the DRC. However, that would add extra data that is not required in this case to see how to fix the spacing error.

At this scale, it is easy to see that the contact is $\frac{1}{2}\mu\text{m}$ too close to the edge of the gate. Now we need to move it $\frac{1}{2}\mu\text{m}$ to the left. However, if you try to select the just the contact, you will not be successful. This is because it is contained in a nested cell. To edit the cell with the error, the easiest method is to use the PEDIT command. Type the following at the command prompt:

PEDIT NEAR

Now click on the edge of the contact where it is not covered by the error wire so there is no conflict. You are now editing the nested cell ADVCKT. Verify this by noting the “ADVTUTR:ADVCKT” message at the top of the window.

Now you can select the contact with the following command:

SELECT IN

The
UNSELECT
ALL command
is used to insure
that no
components are
selected.

Using the cursor, define a box that intersects an edge of the contact (as shown in Figure 222) and then make sure that only one component is selected. The message on the prompt line should end with "Sel=1". Now move the component and unselect it with the commands:

MOVE -.5, 0
UNSEL ALL

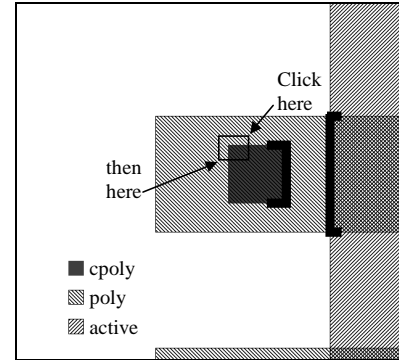


Figure 222: Selecting contact.

Now go back to the main cell, while saving your change to the subcell, with the following command:

EXIT

Now that you are back at the main cell, type (or select from the menu) the VIEW ALL command to see the entire cell. By zooming in on the other selected error marks with the VIEW BOX command, you can see that editing the ADVCKT cell has fixed 3 of the 4 errors marked on layer 101.

The cell on the lower right is a different cell and needs to be fixed separately. **Follow the same steps beginning on the previous page to fix the error in the lower right cell, ADVCKTP.**

Once back at the main cell, you can delete the error marks. If the same 8 error wires are still selected, simply type the DELETE command. If other shapes were selected, or if you unselected the error wires, type the following commands:

UNSELECT ALL
SELECT LAYER 101 ALL
DELETE

Now export the design again for DRC checking with the following command:

DRC

Advanced Tutorial: Simple Spacing Check

You can exit the layout editor session at this point. Execute the DRC again with the command line shown on page 386. The end of the console messages should now look similar to:

```
Done.  
  
100% of chip done.  
***No input skipped.  
***No bad ICED polygons.  
***No errors found  
***Total output non-error output count=0  
    0 total figures output to non-error layers.  
    0 total figures output to error layers.
```

This indicates that no other violations of the contact to gate spacing design rule are present in the layout.

Next we move on to a spacing rule that could be difficult to check without generating a lot of false errors.

Directional Spacing Check

Subjects covered below
Avoiding false MINSPACING errors for overlap extension spacing
Refining MINSPACING rule with directional keywords
Reminder when modified rule set is not recompiled

Many spacing design rules need a more complex check than just a simple spacing check from all shapes on one layer to all shapes on another layer. Take the case of a minimum spacing check from a gate device (all overlaps of active and poly) to the edge of the extension on the poly layer.

You need to prevent false errors that will arise from a simple spacing check. You do not want to mark coincident edges of poly and gate, perpendicular edges, or edges of separate poly shapes.

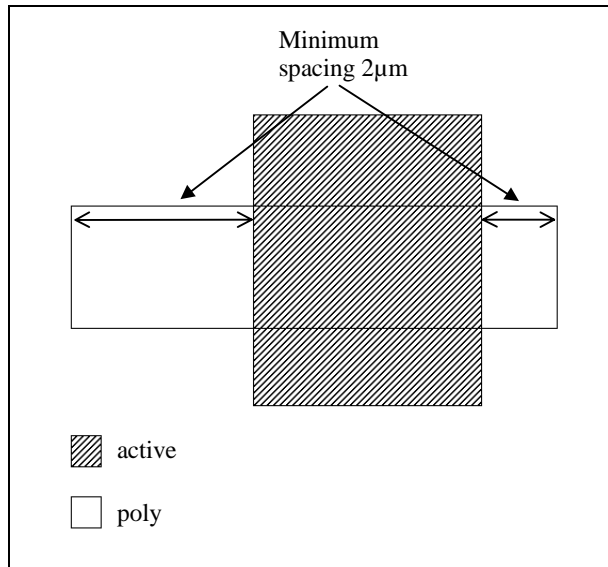


Figure 223: Minimum spacing rule for poly extensions beyond gates.

A simple spacing check will mark false errors for all edges of all gate shapes since all have edges that are coincident with and perpendicular to poly shapes. These false errors are indicated in Figure 224.

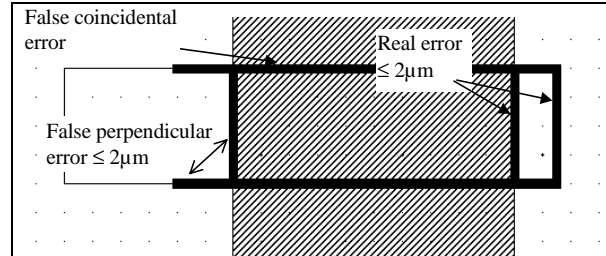


Figure 224: MIN_SPACING (gate, poly, 2)

A better test is to look only outward from the gate shapes for edges of poly shapes. This is accomplished by adding the /OUT keyword after the gate layer name in the rule. This will prevent coincident edges from being marked. Perpendicular edges will also not be marked since the /OUT keyword automatically invokes the /~PERP option.

However, this type of test can still mark a class of false errors. Unrelated poly shapes that are too close may be marked. However, if you see no false errors of the type in Figure 225, simply adding the /OUT keyword to the rule as shown may be your best solution to this type of problem rule.

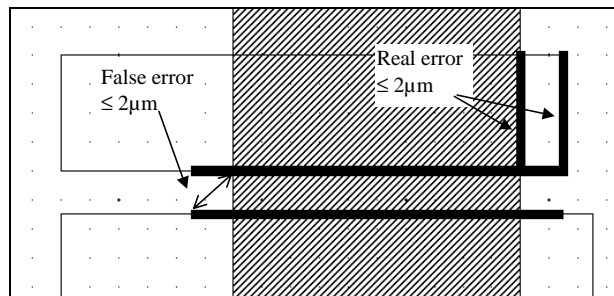
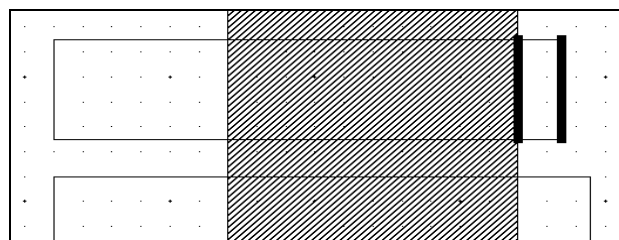


Figure 225: MIN_SPACING (gate/OUT, poly, 2)

Gates with 0 poly extension will not be marked as errors by either rule with the /OUT keyword. We cover how to add this test on page 397.

A better test is to add the /IN keyword after the poly layer to restrict potential errors to those seen looking in from the edge of poly shapes. This will avoid the type of false errors seen in Figure 225. When both /OUT and /IN are used, only the real error shown in Figure 226 is marked.



**Figure 226:
MIN_SPACING (gate/OUT, poly/IN, 2)**

Modifying the Rule Set

We need to add two lines to our rule set to add the poly extension rule. The actual spacing check is created by adding the following line to the end of the rule set:

gate_overlap_err = MIN_SPACING (gate/OUT, poly/IN, 2)

where “gate_overlap_err” is the name of the layer where the error marks will be created. We need to define this layer at the end of the layer definition lines with the following line:

OUTPUT LAYER 102 gate_overlap_err

Error marks will now be created on layer number 102 when the output from the DRC is imported into your cell.

Your rule set should now look like Figure 227. Save the file.

Now let's see what happens if you forget to re-compile the rule set before executing the DRC. Execute the DRC now as shown on page 386.

```
ALL_SAFE
INPUT LAYER 4 active !Diffusion
INPUT LAYER 8 poly !Poly
INPUT LAYER 9 cpoly !Contact from Metal to Poly
OUTPUT LAYER 0 gate !Transistor gate
OUTPUT LAYER 101 gate_cpoly_sp_err
OUTPUT LAYER 102 gate_overlap_err
WIRE_WIDTH=0.3

gate = active AND poly
gate_cpoly_sp_err = MIN_SPACING (gate, cpoly, 2)
gate_overlap_err = MIN_SPACING (gate/OUT, poly/IN, 2)
```

Figure 227: New contents of the ADVTUTR.RUL file.

Advanced Tutorial: Directional Spacing Check

The console messages should end with something like the following:

```
Q:\ICED\ADVTUTR\ADVTUTR.BB was not made with the current
version of the file Q:\ICED\ADVTUTR\ADVTUTR.RUL.
```

```
Q:\ICED\ADVTUTR\ADVTUTR.RUL was created 1 Jan, 2001,
09:00:00
```

```
Q:\ICED\ADVTUTR\ADVTUTR.BB was generated with input file
Q:\ICED\ADVTUTR\ADVTUTR.RUL created 1 Jan 2001, 08:00:00
```

```
Do you want to proceed anyway<Y/[N]>?
```

You must reply to this prompt to proceed. The square brackets around the N imply that simply pressing <Enter> will indicate that you have chosen the default reply of “NO”. Press <Enter> now to terminate the DRC.

To run the modified rules, we first need to recompile the rule set. Use the same method used on page 385.

The DRC will remind you to recompile the rules when you forget, but it cannot remind you when you have forgotten to recreate the binary layout data file. Any cell file may be have changed. Checking the date stamp of the main cell is not enough to insure that the layout has not changed. You must be sure to always recreate the binary layout data file with the DRC command from the main cell in the layout editor any time you change the layout.

Now execute the DRC as shown on page 386. The end of the console messages should now look similar to the following:

```
Done.
```

```
100% of chip done.
***No input skipped.
***No bad ICED polygons.
***No acute angles were output
***Error count=2 (raw=3)
***Total output non-error output count=0
    0 total figures output to non-error layers.
    2 total figures output to error layers.
```

Open the layout editor. Import the DRC generated shapes with the following command on the editor command prompt line:

@DRCOUT

To select the error shapes and see them clearly, type the following commands:

**UNSELECT ALL
SELECT LAYER 102 ALL
VIEW SEL
VIEW OUT 6**

If your error wires extend past the edges, execute the USE WIRETYPE=0 command before importing the DRC shapes.

You should now see two error marks indicating an error as shown in Figure 228.

Once again, the error is contained in a nested cell. To edit the cell with the error, use the PEDIT command. Type the following at the editor command prompt:

PEDIT NEAR

Now click on an edge of the poly shape in error as shown in Figure 228.

Now you are editing the subcell ADVCKTP. You need to select only one side of the poly shape to stretch it rather than shift the entire shape. Select the only the end side by using the following command and selecting the correct side as shown in Figure 229 on the next page. The error wire will not interfere with selecting the side since the error wires are contained in the main cell, not the cell you are currently editing.

SELECT SIDE IN

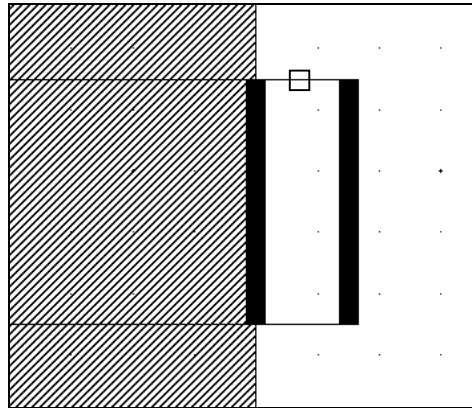


Figure 228: Selecting shape in cell

You should see a single selection mark on the one side. Now you can stretch the poly shape by using the MOVE command. First make sure that only one component is selected. The message on the prompt line should end with “Sel=1”. Now move the component and unselect it with the commands:

```
MOVE .5, 0  
UNSEL ALL
```

Now save your change to the subcell and return to the main cell with the following command:

```
EXIT
```

Now that you are back at the main cell, delete the error marks. If the same 2 error wires are still selected, simply type the DELETE command. If other shapes were selected, or if you unselected the error wires, type the following commands:

```
UNSELECT ALL  
SELECT LAYER 102 ALL  
DELETE
```

Now export the design again for DRC checking with the following command:

```
DRC
```

Run the DRC as shown on page 386. The console messages should now report that no errors were found.

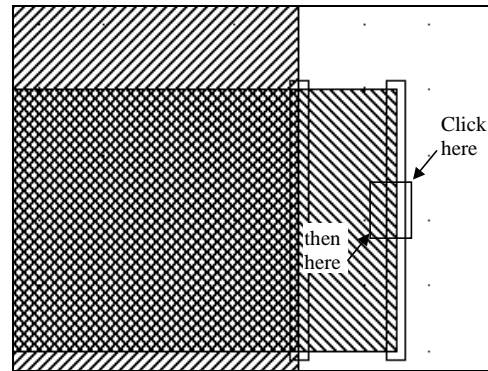


Figure 229: Selecting end of poly shape.

Finding Errors Involving Touching Shapes

Subjects covered below
Finding coincident edge errors not found by MINSPACING rule
TOUCHING rule
Counting shapes as errors that would not ordinarily be counted as errors

When you need to add directional keywords to a spacing rule as we did in the last example, there is an important side effect of which you should be aware. Coincident edges are excluded from consideration when directional keywords are added to a MINSPACING rule.

This is usually desirable. However it can prevent real spacing errors formed by a 0 overlap from being marked.

In the previous example that tested the minimum spacing of poly extensions beyond gate shapes, you may have noticed that one gate had no poly extension. See Figure 230. However this gate was not marked as an error. If all coincident poly-gate edges had been marked, many false errors would have been marked. How do we add a rule so that only errors like the one above are found?

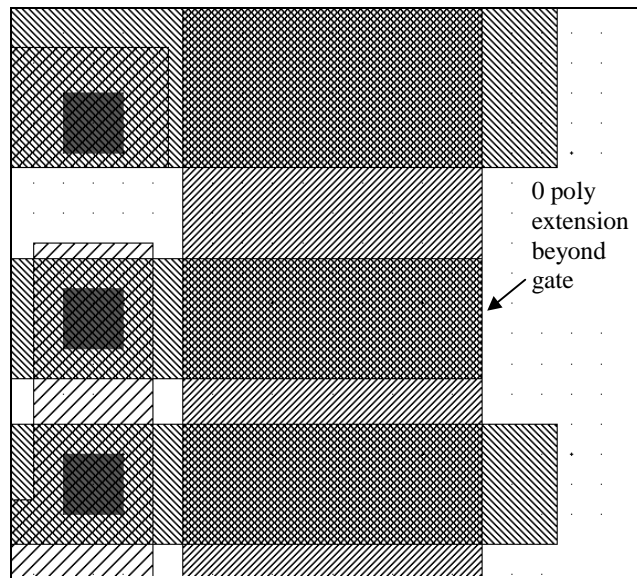


Figure 230: Coincident poly-gate edge that does form an error.

Several solutions are possible for this particular example, including removing the directional keywords from the MINSPACING rule, and then filtering the error shapes to get rid of the false errors. However the best solution is to find all gate shape edges that do not have extensions of either poly or active on every side. This way we also find gate shapes with no active extension as shown in Figure 231.

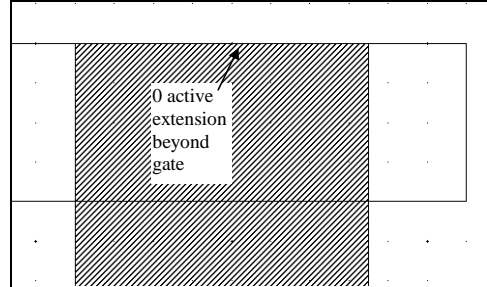


Figure 231: Coincident active-gate edge that does form an error.

The only way to test that every side of each gate shape has a non-zero overlapping shape of either poly or active is a TOUCHING test. We need to test that no gate shape touches a layer formed by the inverse of the union of the poly and active layers.

First let us add the lines that will create this inverse layer. Add these lines to the end of the rules file.

```
gate_overlaps = poly OR active  
not_gate_overlaps = NOT gate_overlaps
```

Now add a TOUCHING rule to find gates that touch this inverse layer and copy them to an error layer. Add the following line to end of the rules file.

```
gate_no_overlap_err = gate TOUCHING not_gate_overlaps
```

Note that we stated above that the shapes should be copied to an error layer. The results of the TOUCHING rule are not automatically counted as errors. If we do not define the gate_no_overlap_err layer as an error layer, the shapes created on it would not be added to the error count and could easily be missed.

We need to define the layer with the ERROR keyword in the OUTPUT LAYER rule.

```
OUTPUT ERROR LAYER 103 gate_no_overlap_err
```


Our other error layers are the result of MINSPACING rules, and the result layers of MINSPACING rules are automatically considered error layers. You do not need to add the ERROR keyword to these rules. However the layers created by TOUCHING rules have other uses than finding errors, so the DRC does not automatically consider these layers as error layers. Any layer may be defined as an error layer, including the results of Boolean processing or size/shape rules such as ASPECT_RATIO or IS_BOX.

The scratch layers also need to be defined. Add the following lines:

OUTPUT LAYER 0 gate_overlaps
OUTPUT LAYER 0 not_gate_overlaps

```
ALL_SAFE

INPUT LAYER 4 active !Diffusion
INPUT LAYER 8 poly !Poly
INPUT LAYER 9 cpoly !Contact from Metal to Poly

OUTPUT LAYER 0 gate !Transistor gate
OUTPUT LAYER 0 gate_overlaps
OUTPUT LAYER 0 not_gate_overlaps

OUTPUT LAYER 101 gate_cpoly_sp_err
OUTPUT LAYER 102 gate_overlap_err
OUTPUT ERROR LAYER 103 gate_no_overlap_err
WIRE_WIDTH=0.3

gate = active AND poly

gate_cpoly_sp_err = MIN_SPACING (gate, cpoly, 2)

gate_overlap_err = MIN_SPACING (gate/OUT, poly/IN, 2)

gate_overlaps = poly OR active
not_gate_overlaps = NOT gate_overlaps
gate_no_overlap_err = gate TOUCHING not_gate_overlaps
```

Figure 232: New contents of the ADVTUTR.RUL file.

Advanced Tutorial: Directional Spacing Check

Save the rules file and compile it. Run the DRC. The console messages should indicate that 1 shape was created on an error layer.

Open the layout editor. Import the DRC results once the editor is open.

@DRCOUT

To select the error shapes and see them clearly, type the following commands:

**UNSELECT ALL
SELECT LAYER 103 ALL
VIEW SEL
VIEW OUT 6**

You should now see the gate shown in Figure 230 on page 397 has been copied to the error layer 103. Delete the shape on layer 103 with the following command:

DELETE

This time the shapes that form the transistor are not nested in a subcell. You can select the correct side of the poly shape with the following command:

SELECT LAYER POLY SIDE IN

You must specify the poly layer in the above selection rule otherwise the shape on active would be selected at the same time. Now move the poly edge to the correct minimum overlap distance with the command:

MOVE 2, 0

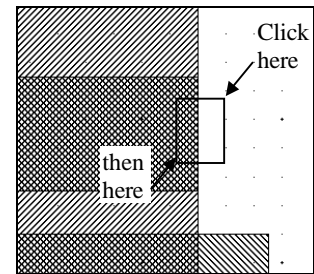


Figure 233: Selecting end of poly shape.

Now recreate the layout file for the next exercise with the DRC command and exit the editor. Rerun the DRC if you desire to see that the design is now error-free.

Our rule set now contains an adequate check for minimum poly shape extension beyond gate shapes. You may wonder about the case where a poly shape does not even extend to the edge of the active shape as shown in Figure 234.

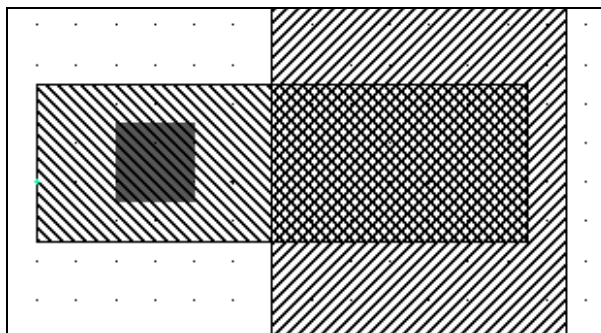


Figure 234: Invalid gate

While this type of error will not be found by this DRC rule set, it will still be found. Any program that recognizes devices and circuits to compare the layout to a schematic will short the source of the gate to the drain and will report the error.

However, such circuit discrepancies can be difficult to locate. If you prefer to find these types of errors with the DRC, you can add the following rules (and the appropriate layer definitions) to the rule set.

source_drain = active AND NOT poly
bad_gate = gate NOT TOUCHING 2 source_drain

This will copy to the bad_gate layer all gate shapes that do not touch two separate source/drain shapes. Be sure to define the bad_gate layer as an error layer.

Tests That Involve Electrical Connections

Subjects covered below
Defining electrical connections
Masking input layers into 2 different layers with the TOUCHING rule
Adding /~CONN to MINSPACING rules
Verifying wells with the STAMP rule

Many spacing rules involve electrical connections. For example, let us suppose that metal2 shapes that are part of a pad construct must be at least 30 μm away from unconnected shapes on the same layer. (In this case, unconnected indicates an absence of an electrical connection.)

If we tested that all metal2 shapes must be at least 30 μm apart, probably most of the metal2 shapes in a typical chip would be marked as false errors. We must identify the metal2 shapes that are in pad constructs and treat them differently from the rest of the layer. We must also avoid marking as false errors metal2 shapes that are close to the pad, but are electrically connected.

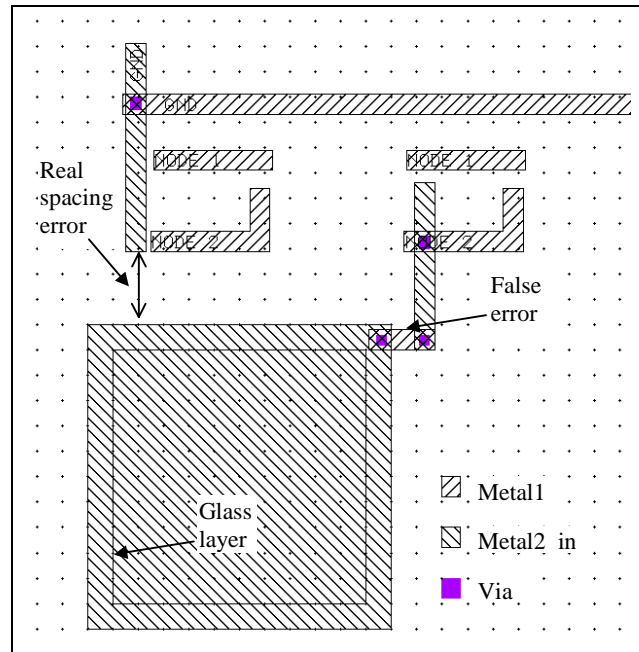


Figure 235: Pad and metal wiring

Masking the Metal2 Layer

The IS_BOX rule with exact dimensions could also be used to find pad metal. However touching shapes on the same layer are merged by the DRC during preprocessing, so some pads may not be truly rectangular and will not be collected by the IS_BOX rule.

We can use various methods to differentiate the metal2_in layer. If your pad constructs are contained in separate cells you could use the INCELL keyword in the INPUT LAYER rule to separate the metal2_in layer. However, let us assume that the pad shapes are either in the main cell, or in cells with many different names. Then the easiest method to find pad metal is to use proximity to the overglass layer.

We could use a simple Boolean rule such as the following, but if the glass layer has a different outline than the metal layer, the entire metal2 shape that forms the pad will not be moved to the pad_metal2 layer, only that portion that is covered by the glass layer. (See Figure 235.) The spacing rules will measure to this boundary rather than the larger metal2 shape that really represents the pad.

pad_metal2 = metal2_in AND glass !will identify only parts of pad

A better method is to consider all metal2 that touches a glass shape as pad_metal2. For this we need a TOUCHING rule. Add the following rules before the first MIN_SPACING rule.

pad_metal2 = metal2_in TOUCHING glass
metal2 = metal2_in AND NOT pad_metal2

Now the metal2_in layer will be separated into two non-overlapping layers: metal2 and pad_metal2. The entire metal2_in shape that comprises the pad construct will be moved to layer pad_metal2.

We will need to define the input layers for this processing. Add the following lines to the end of the INPUT LAYER rules:

INPUT LAYER 11 metal1	!First Metal
INPUT LAYER 12 via	!Contact from Metal1 to Metal2
INPUT LAYER 13 metal2_in	!Second Metal
INPUT LAYER 16 glass	!Pad OverGlass layer

We also need to define the new scratch layers and the new error layer. Add the following lines to appropriate places:

```
OUTPUT LAYER 0 metal2
OUTPUT LAYER 0 pad_metal2

OUTPUT ERROR LAYER 104 m2_pad_spacing_err
```

Adding Electrical Connection Rules

Electrical connections are defined with the CONNECT rule. This rule can connect two layers by a contact or via layer, or directly connect two layers. We need both forms to implement the electrical connections for this test.

If you use the NLE program to extract the circuit data from the layout, you can create the CONNECT rules in a separate file and include the file in both the DRC and NLE rule sets with the INCLUDE rule. This insures that the circuit recognition rules are identical and up to date for both programs.

```
CONNECT metal2      metal1 BY via
CONNECT pad_metal2  metal1 BY via
CONNECT metal2      pad_metal2
```

The last 2 CONNECT rules are important. Whenever you mask a layer into two separate layers, you must remember to account for both new layers in the CONNECT rules. Duplicate the CONNECT rules involving contacts or vias. Add a CONNECT rule to connect the two layers together.

The placement of CONNECT rules is important. They must occur after the layer manipulation rules, because layer manipulation may invalidate the previous electrical connections. The CONNECT rules must occur before any MINSPACING rules since the electrical connections may invalidate the results of previous spacing rules that checked connectivity. This rule order is enforced by the compiler. The error messages can vary based on the contents of the rule set, so just be sure to divide your rule set as shown in Figure 236.

Safe processing directive
Layer definition rules
Layer manipulation
Electrical connections
Spacing and other verification rules

Figure 236: Rule set order

In our case, add the CONNECT rules just before the first MIN_SPACING rule.

Finally, we need to add the MIN_SPACING rule that tests the pad spacing. Add the following rule directly after the CONNECT rules:

m2_pad_spacing_err = MINSPACING(metal2, pad_metal2, 30 /~CONN)

The /~CONN option will prevent the DRC from marking errors on shapes that are electrically connected even if they are closer than 30 microns. All unconnected shapes will be marked as errors if they are too close.

The entire rule set now should look like Figure 237 on the next page.

You will probably have other spacing rules that involve the M2 layer. Be careful to test both the metal2 and pad_metal2 layers for each of these rules, just as you had to add CONNECT rules for both layers.

For example, a test for metal2 overlap of the via layer should test both metal2 layers.

```
m2_via_overlaperr = MINSPACING(via/OUT, metal2/IN, 1)
padm2_via_overlaperr = MINSPACING(via/OUT, pad_metal2/IN, 1)
```

However, in this case, since no electrical connections are involved, you can test both layers with the original undifferentiated layer.

```
m2_via_overlaperr = MINSPACING(via/OUT, metal2_in/IN, 1)
```

If electrical connections were involved, you would have to test both layers individually since electrical connections are not defined for the metal2_in layer. No shapes on metal2_in are electrically connected to any other shape.

```

ALL_SAFE

INPUT LAYER 4 active      !Diffusion
INPUT LAYER 8 poly       !Poly
INPUT LAYER 9 cpoly      !Contact from Metal to Poly
INPUT LAYER 11 metal1    !First Metal
INPUT LAYER 12 via       !Contact from Metal 1 to Metal 2
INPUT LAYER 13 metal2_in !Second Metal
INPUT LAYER 16 glass     !Pad OverGlass layer

OUTPUT LAYER 0 gate      !Transistor gate
OUTPUT LAYER 0 gate_overlaps
OUTPUT LAYER 0 not_gate_overlaps
OUTPUT LAYER 0 metal2
OUTPUT LAYER 0 pad_metal2

OUTPUT LAYER          101 gate_cpoly_sp_err
OUTPUT LAYER          102 gate_overlap_err
OUTPUT ERROR LAYER 103 gate_no_overlap_err
OUTPUT ERROR LAYER 104 m2_pad_spacing_err
WIRE_WIDTH=0.3

gate = active AND poly

pad_metal2 = metal2_in TOUCHING glass
metal2 =      metal2_in AND NOT pad_metal2

CONNECT metal2      metal1 BY via
CONNECT pad_metal2  metal1 BY via
CONNECT metal2      pad_metal2

m2_pad_spacing_err = MIN_SPACING(metal2, pad_metal2, 30 /~CONN)
gate_cpoly_sp_err =  MIN_SPACING (gate, cpoly, 2)
gate_overlap_err =   MIN_SPACING (gate/OUT, poly/IN, 2)

gate_overlaps =      poly OR active
not_gate_overlaps =  NOT gate_overlaps
gate_no_overlap_err = gate TOUCHING not_gate_overlaps
    
```

Figure 237: New contents of the ADVTUTR.RUL file.

Looking at the Pad Spacing Error

Save the rules file and compile it. We do not need to regenerate the layout file since the layout has not changed. Run the DRC. The console messages should indicate that 2 shapes were created on an error layer.

Open the layout editor and import the DRC results.

@DRCOUT

The error marks should look similar to Figure 238. Note that the DRC marked only the true error where the metal2 that forms the ground wire comes too close to the pad. The false error of the connected metal2 wire on the right did not get marked.

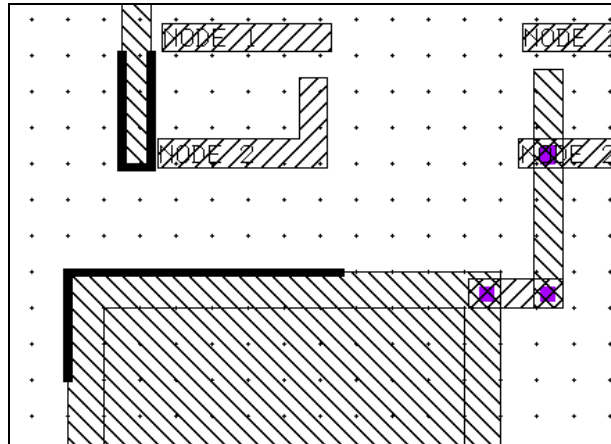


Figure 238: Error marks for pad rule

See page 400 to learn how to select a shape on a certain layer even when it overlaps shapes on other layers.

Select the end of the metal2 ground wire in error where it is marked with the error shape. This wire is in the main cell. Now shift the end of the wire above the extent of the error marks. Delete the error marks on layer 104, and regenerate the layout file with the DRC command. Exit the cell to save your changes.

Adding the Rest of the Good Conductor Electrical Connection Rules

Let us assume that we have many rules that depend on electrical connections. We need to define the electrical connections for all conductive layers on the chip. In addition, we need to define the connectivity for poor conductor layers, such as wells. We will do a poor conductor layer in a separate step beginning on page 411.

The entire active layer is not a conductive layer. We must mask it and create the source/drain layer by removing the transistor gates before we can use it in the connect rules. Add the following rule directly below the “gate = active AND poly” rule.

srdrn = active AND NOT poly

We need to define this new scratch layer with the following rule:

OUTPUT LAYER 0 srdrn !Source/Drain layer with gates removed

We need to add a new input layer that forms the connections between the active layer and the metal1 layer.

INPUT LAYER 10 cactive !Contact from Metal 1 to Diffusion

Now we are ready for the additional CONNECT rules.

CONNECT srdrn metal1 BY cactive
CONNECT poly metal1 BY cpoly

The entire rule set now should look like Figure 239 on the next page.

```

ALL_SAFE

INPUT LAYER 4 active      !Diffusion
INPUT LAYER 8 poly       !Poly
INPUT LAYER 9 cpoly      !Contact from Metal to Poly
INPUT LAYER 10 cactive    !Contact from Metal to Diffusion
INPUT LAYER 11 metal1     !First Metal
INPUT LAYER 12 via        !Contact from Metal 1 to Metal 2
INPUT LAYER 13 metal2_in  !Second Metal
INPUT LAYER 16 glass      !Pad OverGlass layer

OUTPUT LAYER 0 gate       !Transistor gate
OUTPUT LAYER 0 srcdrn     !Source/Drain layer with gates removed
OUTPUT LAYER 0 gate_overlaps
OUTPUT LAYER 0 not_gate_overlaps
OUTPUT LAYER 0 metal2
OUTPUT LAYER 0 pad_metal2

OUTPUT LAYER 101 gate_cpoly_sp_err
OUTPUT LAYER 102 gate_overlap_err
OUTPUT ERROR LAYER 103 gate_no_overlap_err
OUTPUT ERROR LAYER 104 m2_pad_spacing_err
WIRE_WIDTH=0.3

gate = active AND poly
srcdrn = active AND NOT poly

pad_metal2 = metal2_in TOUCHING glass
metal2 = metal2_in AND NOT pad_metal2

CONNECT metal2 metal1 BY via
CONNECT pad_metal2 metal1 BY via
CONNECT metal2 pad_metal2
CONNECT srcdrn metal1 BY cactive
CONNECT poly metal1 BY cpoly

m2_pad_spacing_err = MIN_SPACING(metal2, pad_metal2, 30 /~CONN)
gate_cpoly_sp_err = MIN_SPACING (gate, cpoly, 2)
gate_overlap_err = MIN_SPACING (gate/OUT, poly/IN, 2)

gate_overlaps = poly OR active
not_gate_overlaps = NOT gate_overlaps
gate_no_overlap_err = gate TOUCHING not_gate_overlaps

```

Figure 239: New contents of the ADVTUTR.RUL file.

Now we can add other verification rules that involve connectivity. Let us assume that the minimum distance from a poly contact to an unconnected poly shape is 4 microns. (See Figure 240.) If we tested the spacing between all contacts and the poly layer, almost every contact would be marked as a false error since all are covered by poly. We need to add the `/~CONN` option to the `MIN_SPACING` rule to avoid all of these false errors. Add the following with the other spacing rules:

`poly_unconn_contact_err = MINSPACING(poly, cpoly, 4 /~CONN)`

We also need to define this new error layer with the other error layers.

OUTPUT ERROR LAYER 105 poly_unconn_contact_err

Now save the file and compile it. Note that at the end of the console messages, the following comment is present:

Connected layers form
one group

See an example
of an
incomplete rule
set that creates 2
groups on 413.

This message is important. If you see a different message stating that 2 or more separate groups have been formed, check the log file (DRCOUT.DLO in our case) to see details on each group to help figure out if a `CONNECT` rule is missing or mistyped.

Run the DRC. The console messages should report that 2 error marks have been created. When you open the layout editor and import the DRC results, you should see that only the top contact shown in Figure 240 has been marked. Move the top contact (in the main cell) up 0.5 units to fix the error.

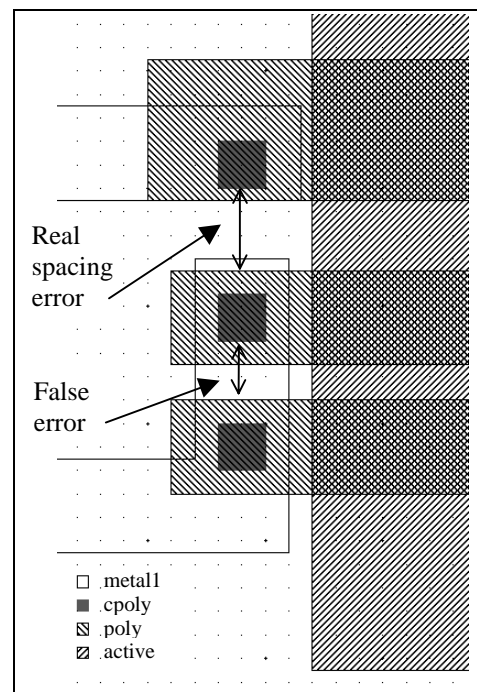


Figure 240: Poly contacts

The bottom contacts are not in error since they are connected electrically to the near pieces of poly wire.

NWELL Connections and Verification

Let us suppose that some of your verification rules involve testing the spacing between unconnected NWELL shapes. To test this, we need to extend the definition of electrical connections to include wells.

Well layers represent poor conductors, rather than good conductor layers such as metal or poly. Good conductors can connect to them, but connections should not pass through them. We can demonstrate the importance of verifying well connections with Figure 241. Let us assume that the GND wire on the right connects to the metal GND bus and from there to a pad on the chip. However, the GND wire on the left does not connect to the bus. You meant to connect these two wires, but a gap exists by accident. The ground wire on the left is not really electrically connected to the wire on the right. If you used only CONNECT rules to define electrical connections to wells, then the problem would not be found.

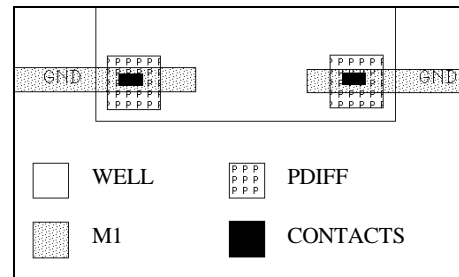


Figure 241: Open on GND node that connects only through WELL layer.

This is mainly a circuit recognition problem, but if you use CONNECT rules to define the electrical connections for well shapes, you may prevent real spacing verification errors involving electrical connections from being found. False errors may be generated. It is best to find electrical shorts or opens to well shapes early in the design process, rather than waiting for circuit recognition and comparison (LVS) tests.

The STAMP rule is the best method for defining electrical connections to poor conductor layers. The STAMP rule allows connections to poor conductors, but does not allow connections to pass through them to other nodes. The well in

Figure 241 would be marked as an overstamped well, i.e. a poor conductor that connects to two different electrical nodes.

Let us assume that the connections to the nwell layer in our example are formed as follows:

A shape on the active layer that is covered by a shape on the nselect layer and a shape on the nwell layer forms a shape on the nplus layer. The nplus layer connects the nwell shape to the active shape.

To generate the nplus layer, add the following lines just below the “sdrdn = active AND NOT poly” rule:

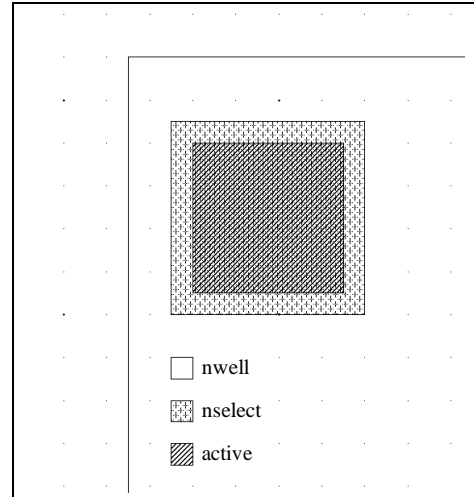


Figure 242: Nwell contact

```
welldiff = sdrdn AND nwell
nplus = welldiff AND nselect
```

These new input layers and scratch layers must be defined as shown below:

```
INPUT LAYER 3 nwell_in !As-drawn Nwell
INPUT LAYER 7 nselect !N-select

OUTPUT LAYER 0 nplus !N+
OUTPUT LAYER 0 welldiff !Nwell diffusion
OUTPUT LAYER 0 nwell !Generated Nwell
```

In a later lesson, we will be generating the nwell layer for export based on the as-drawn nwell layer and some other layer interaction. For now, we can test these rules using the nwell layer as it is drawn. So add the following line just after the layer definition rules, above all other layer processing rules:

```
nwell = nwell_in !temporary nwell layer generation
```

The rule to form the electrical connections follows. Add this rule just after the CONNECT rules:

STAMP nwell BY nplus MULTI=nwell_shorts NONE=nwell_opens

This STAMP rule can create shapes on two new error layers: nwell_shorts and nwell_opens. Define these new error layers near the other error layer definitions.

OUTPUT ERROR LAYER 106 nwell_shorts
OUTPUT ERROR LAYER 107 nwell_opens

The nwell_shorts layer will contain copies of all nwell shapes that connect to two or more separate electrical nodes. The nwell_opens layer will contain copies of all nwell shapes that have no connection to any other electrical node, i.e. unconnected wells.

If we saved the rules and compiled them at this point, we would have a problem. The rules compiler would post the following message to the console:

```
*****WARNING**Connected layers form 2 groups.
```

To determine the source of this problem, we would need to look at the rules compiler log file. It would contain the following lines near the end of the file:

```
*****WARNING**Connected layers form 2 groups.
Connection group 1:
  POLY[8]
  CPOLY[9]
  CACTIVE[10]
  METAL1[11]
  VIA[12]
  SRCDRN
  METAL2
  PAD_METAL2

Connection group 2:
  NPLUS
  NWELL
```

Looking at the two lists above, it is easy to see that we forgot to connect the nplus layer to the rest of the connected layers. The nplus layer connects to srcdrn layer. It can be easy to forget that even though one layer is generated from another, the electrical connections are not inherited in any fashion.

The compiler message is just a warning. We could go ahead and run the DRC with the compiled rules that generated the warning. If we did, every nwell shape would be copied to the nwell_opens layer.

To solve the problem, we need one more CONNECT rule. Add the following rule just above the STAMP rule:

CONNECT srcdrn nplus

Now we can add a rule to test the spacing of unconnected well shapes. Let us assume that well shapes that are not electrically connected to each other must be at least 10 microns apart. The following rule will test this without marking false errors for well shapes that are electrically connected.

nwell_sp_err = MINSPACING(nwell, nwell, 10 /~CONN)

The layer definition rule for this new error layer must be included.

OUTPUT ERROR LAYER 108 nwell_sp_err

Now the entire rule set looks as follows:

ALL_SAFE

```

INPUT LAYER 3 nwell_in    !As-drawn Nwell
INPUT LAYER 4 active     !Diffusion
INPUT LAYER 7 nselect    !N-select
INPUT LAYER 8 poly       !Poly
INPUT LAYER 9 cpoly      !Contact from Metal to Poly
INPUT LAYER 10 cactive   !Contact from Metal to Diffusion
INPUT LAYER 11 metal1    !First Metal
INPUT LAYER 12 via       !Contact from Metal 1 to Metal 2
INPUT LAYER 13 metal2_in !Second Metal
INPUT LAYER 16 glass     !Pad OverGlass layer

OUTPUT LAYER 0 gate      !Transistor gate
OUTPUT LAYER 0 srcdrn    !Source/Drain layer with gates removed
OUTPUT LAYER 0 gate_overlaps
OUTPUT LAYER 0 not_gate_overlaps
OUTPUT LAYER 0 metal2
OUTPUT LAYER 0 pad_metal2
OUTPUT LAYER 0 nplus     !N+
OUTPUT LAYER 0 welldiff  !Nwell diffusion
OUTPUT LAYER 0 nwell     !Generated Nwell

OUTPUT LAYER 101 gate_cpoly_sp_err
OUTPUT LAYER 102 gate_overlap_err
OUTPUT ERROR LAYER 103 gate_no_overlap_err
OUTPUT ERROR LAYER 104 m2_pad_spacing_err
OUTPUT ERROR LAYER 105 poly_unconn_contact_err
OUTPUT ERROR LAYER 106 nwell_shorts
OUTPUT ERROR LAYER 107 nwell_opens
OUTPUT ERROR LAYER 108 nwell_sp_err
WIRE_WIDTH=0.3

```

```
nwell = nwell_in

gate    = active AND poly
srcdrn  = active AND NOT poly
welldiff = srcdrn AND nwell
nplus   = welldiff AND nselect

pad_metal2 = metal2_in TOUCHING glass
metal2 = metal2_in AND NOT pad_metal2

CONNECT metal2 metal1 BY via
CONNECT pad_metal2 metal1 BY via
CONNECT metal2 pad_metal2
CONNECT srcdrn metal1 BY cactive
CONNECT poly metal1 BY cpoly
CONNECT srcdrn nplus
STAMP    nwell BY nplus MULTI=nwell_shorts NONE=nwell_opens

m2_pad_spacing_err =      MINSPACING(metal2, pad_metal2, 30 /~CONN)

nwell_sp_err =             MINSPACING(nwell, nwell, 10 /~CONN)
gate_cpoly_sp_err =        MIN_SPACING (gate, cpoly, 2)
poly_unconn_contact_err =  MIN_SPACING(poly, cpoly, 4 /~CONN)
gate_overlap_err =         MIN_SPACING (gate/OUT, poly/IN, 2)

gate_overlaps = poly OR active
not_gate_overlaps = NOT gate_overlaps
gate_no_overlap_err = gate TOUCHING not_gate_overlaps
```

Figure 243: New contents of the ADVTUTR.RUL file.

Compile the rule set and run the DRC. The DRC should report that 4 error shapes have been created. If you open the log file you can see what layers these error marks were created on:

```
1 figures output to layer NWELL_OPENS[107]
3 figures output to layer NWELL_SP_ERR[108]
4 total figures output to error layers.
```

Open the layout editor. When you import the results of the DRC with the @DRCOUT command, the results should look similar to Figure 244. Note that the 3 shapes on layer 108 mark two pairs of unconnected wells that are closer than 10 microns. At first glance you may think that the top pair of wells is marked in error. However, once you see that the well on the left has been marked on layer 107 as an unconnected well, you can see that the two wells are really unconnected. No well contact was added to connect the ground wire to the well.

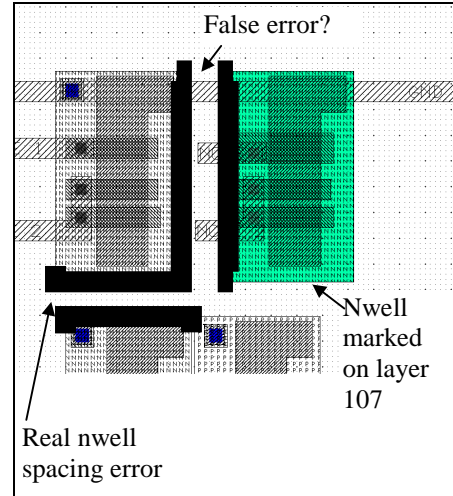


Figure 244: Generated nwell shapes cut at panel boundaries.

This problem must be fixed by adding shapes on layers active, cact, and nselect to the upper left of the nwell shape under the ground wire. You can copy the 3 shapes that form the nwell contact from the guard ring structure below. Do this now.

The lower nwell in Figure 244 could be meant to connect to ground as well. It is not marked as an unconnected well since the contact is present. The STAMP rule tests only that each well is electrically connected to exactly one other node. It does not test that each well is connected to the right node. This type of error needs to be found by the LVS.

Let us assume that the well is meant to connect to a different node. We will fix the problem by shifting both lower cells further away from the other wells. Select the two lower cells and shift them down with the MOVE command.

Recreate the binary layout data file with the DRC command and rerun the DRC to insure that the problems are fixed.

Creation of Shapes for Export

Subjects covered below
Generation of mask layer for export back into cell
Fixing acute angles in generated layer
Finding bad polygons in input layer
Use of the subcell error command file

Sometimes a required mask layer may be difficult to create by hand in the layout editor, but easy to create from other layers with a few simple rules in the DRC. Perhaps a design rule change after much of a chip has been laid out requires simple but widespread changes to a layer.

The process to generate a layer that represents a real mask layer is the same as that for any other output layer. The layer must be defined with a proper layer number.

If you are generating a layer that will replace an old layer, be careful to use an output layer number different from the old layer number in the cell. Keep the layers separate to allow for changes in the layer generation until you have used the new layer extensively. You can swap layer numbers later, or change the Stream or CIF definitions to use the new layer number.

Even if you want to wait until the design is almost finished to import the final version of the new layer, it is best to create the layer in your regular rule set so that any design rule problems between it and the other layers are found early on.

Nwell Layer Generation

For our tutorial, let us say that you discover that an extension of nwell beyond active layer shapes improves reliability. You want to change the nwell layer so that there is a 2.5µm extension of nwell beyond all active layer shapes. However, most of your next chip is already laid out. To fix all nwell shapes by hand might take a considerable amount of time.

You can accomplish the same task with the DRC in a matter of hours. Moreover, you can insure that the new layer is created without design errors, or automatically choose the best option when the optimal design rules cannot be followed in a particularly dense area.

Let us assume that our design includes pwells in addition to nwells. It is critical that the wells do not overlap. This is more important than the extension of the nwell layer beyond the active layer. So we will write the rules to insure that the new nwell layer never overlaps the pwell layer.

First we need to expand the nwell layer to insure the 2.5µm extension beyond the active layer. We do not want to involve any active layer shapes that are not covered by nwell, such as those covered by pwell. We want to use only p_active shapes, those active shapes covered by nwell, for this processing.

Replace the “nwell = nwell_in” rule with the following rules:

```
p_active =      active AND nwell_in
p_active_bloat = BLOAT (p_active, 2.5)
nwell =         nwell_in OR p_active_bloat
nwell =         nwell AND NOT pwell_in
```

This is the first time we have used the pwell layer, so we must now define it in the list of input layers.

```
INPUT LAYER 2 pwell_in      !As-drawn Pwell
```

We need to define the new scratch layers.

```
OUTPUT LAYER 0 p_active  
OUTPUT LAYER 0 p_active_bloat
```

We also need to change the nwell layer number in the output layer definition to a non-zero number so that shapes on the layer are created in the output file. So the “OUTPUT LAYER 0 nwell” rule gets changed to:

```
OUTPUT LAYER 61 nwell           !Generated Nwell
```

So far, we have not used panel rules to control how the layout is divided into chunks for more efficient processing. These rules are not really necessary in our testcase because the default processing is usually adequate. However, a typical chip will be divided into many panels and we want to force those panel boundaries to be in a specific place in our testcase to highlight a possible problem with generated layers.

Add the following lines directly after the ALL_SAFE rule near the top of the file:

```
PANELX =50  
PANELY =50
```

The entire rule set should now look like the one shown in Figure 245.

```

ALL_SAFE

PANELX =50
PANELY =50

INPUT LAYER 2 pwell_in      !As-drawn Pwell
INPUT LAYER 3 nwell_in      !As-drawn Nwell
INPUT LAYER 4 active        !Diffusion
INPUT LAYER 7 nselect       !N-select
INPUT LAYER 8 poly          !Poly
INPUT LAYER 9 cpoly         !Contact from Metal to Poly
INPUT LAYER 10 cactive      !Contact from Metal to Diffusion
INPUT LAYER 11 metal1       !First Metal
INPUT LAYER 12 via          !Contact from Metal 1 to Metal 2
INPUT LAYER 13 metal2_in    !Second Metal
INPUT LAYER 16 glass        !Pad OverGlass layer

OUTPUT LAYER 0 gate         !Transistor gate
OUTPUT LAYER 0 srcdrn       !Source/Drain layer with gates removed
OUTPUT LAYER 0 gate_overlaps
OUTPUT LAYER 0 not_gate_overlaps
OUTPUT LAYER 0 metal2
OUTPUT LAYER 0 pad_metal2
OUTPUT LAYER 0 nplus        !N+
OUTPUT LAYER 0 welldiff     !Nwell diffusion
OUTPUT LAYER 0 p_active
OUTPUT LAYER 0 p_active_bloat
OUTPUT LAYER 61 nwell       !Generated Nwell

OUTPUT LAYER 101 gate_cpoly_sp_err
OUTPUT LAYER 102 gate_overlap_err
OUTPUT ERROR LAYER 103 gate_no_overlap_err
OUTPUT ERROR LAYER 104 m2_pad_spacing_err
OUTPUT ERROR LAYER 105 poly_unconn_contact_err
OUTPUT ERROR LAYER 106 nwell_shorts
OUTPUT ERROR LAYER 107 nwell_opens
OUTPUT ERROR LAYER 108 nwell_sp_err
WIRE_WIDTH=0.3

```

```
p_active =          active AND nwell_in
p_active_bloat =    BLOAT (p_active, 2.5)
nwell =            nwell_in OR p_active_bloat
nwell =            nwell AND NOT pwell_in

gate              = active AND poly
srcdrn            = active AND NOT poly
welldiff          = srcdrn AND nwell
nplus             = welldiff AND nselect

pad_metal2 = metal2_in TOUCHING glass
metal2 = metal2_in AND NOT pad_metal2

CONNECT metal2 metal1 BY via
CONNECT pad_metal2 metal1 BY via
CONNECT metal2 pad_metal2
CONNECT srcdrn metal1 BY cactive
CONNECT poly metal1 BY cpoly
CONNECT srcdrn nplus
STAMP  nwell BY nplus MULTI=nwell_shorts NONE=nwell_opens

m2_pad_spacing_err =    MINSPACING(metal2, pad_metal2, 30 /~CONN)

nwell_sp_err =          MINSPACING(nwell, nwell, 10 /~CONN)
gate_cpoly_sp_err =     MIN_SPACING (gate, cpoly, 2)
poly_unconn_contact_err = MIN_SPACING(poly, cpoly, 4 /~CONN)
gate_overlap_err =      MIN_SPACING (gate/OUT, poly/IN, 2)

gate_overlaps = poly OR active
not_gate_overlaps = NOT gate_overlaps
gate_no_overlap_err = gate TOUCHING not_gate_overlaps
```

Figure 245: New contents of the ADVTUTR.RUL file.

Now save the rule set and compile it. Run the DRC.

The console messages should now look as follows:

```
1 bad ICED polygon
***2 acute angles were output
***No Errors found
***Total output non-error output count=20
    20 total figures output to non-error layers.
    0 total figures output to error layers.
```

We will cover what to do about the bad polygon later on page 426. We will investigate the acute angles below. The 20 non-error output shapes are the shapes generated for the new nwell layer. All of the large nwell shapes generated on layer 61 have been cut at panel boundaries as shown in Figure 246.

When you look at the layout, you can see how the original nwell shapes have been slightly expanded around the active layer. The nwell in the middle has had some material removed near the upper right corner to avoid an overlap with the pwell layer.

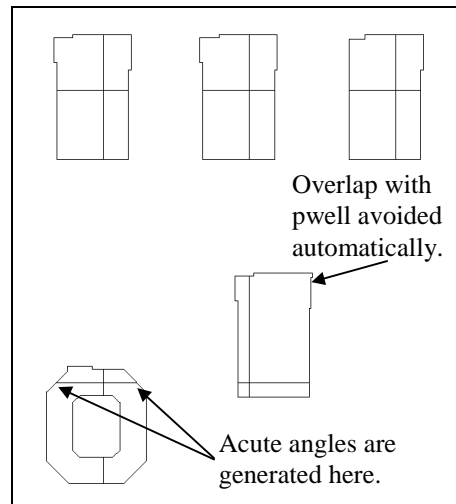


Figure 246: Generated nwell shapes cut at panel boundaries.

Finding the Acute Angles

What causes the acute angle warning when no acute angles were in the input data? Since the guard ring nwell shape has been cut at panel boundaries, the skewed sides near the top have resulted in acute angles in the top two polygons. These types of problems can happen whenever shapes with skewed sides are in the input data used to generate a layer. Sometimes a panel boundary cuts such a shape and the acute angles arise.

Acute angles can be a problem for software that reads mask set data. This can be the case even when the acute angle would disappear when touching shapes are merged. This type of problem is easily fixed in the layout editor. If you will be generating the layer many times as your design progresses, you may choose to ignore these warnings until you are ready to finalize the design and use the generated layer as a mask layer.

The log file will by default list a detailed error message for every acute angle on an output layer. Look at the log file now with your favorite text editor. The log file name is DRCOUT.DLO.

If you want to suppress these acute angle warnings, add the NO_WARN-_ACUTE rule to your rule set. Be sure to remove this rule from the rule set used on your final design.

Near the middle of the log file are the following warning messages:

```
An acute angle was formed on output of ICED
layer 61 at (-66.364, -50)
The acute angle(s) will be marked on layer 99
An acute angle was formed on output of ICED
layer 61 at (-38.636, -50)
```

The error marks created on layer 99 are included with the other shapes created by the DRC. Execute the DRC command file with the @DRCOUT command in the layout editor now.

The acute angles will be marked with error wires on layer 99 as shown in Figure 247. We can easily merge the shapes generated on the new nwell layer (layer number 61) to remove the acute angles. However, remember that the acute angles will be generated every time the DRC is run and you replace layer 61. You may want to wait to fix the acute angles on layer 61 until the design is complete and you begin using layer 61 as your final nwell layer.

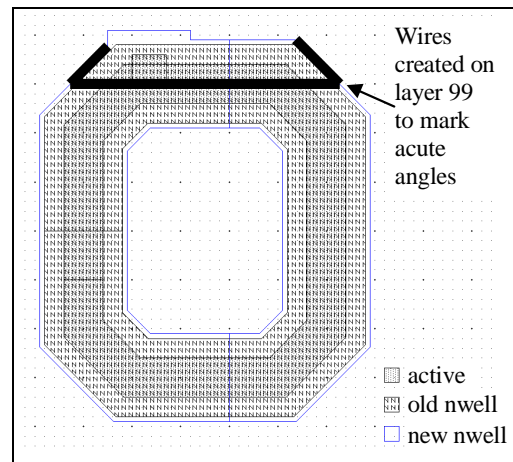


Figure 247: Acute angle marks.

To fix the acute angle on the right, type the following commands, digitizing the select box as shown in Figure 248 during the SELECT command:

**UNSELECT ALL
SELECT LAYER 61 IN
MERGE POLYGONS**

To fix the acute angle on the left, repeat the same three commands, digitizing the select box as shown in Figure 249 during the SELECT command.

There are now no acute angles. However, they will be recreated in the next DRC run unless we change the design to use layer 61 as the real nwell layer and change the rule set to remove the nwell generation rules and change the layer number in the nwell layer definition rule. We will not do this at this time.

The point at which to import DRC generated layers as the real mask layers is dependant on the state of your design and your comfort level with the rules that generate the mask layer.

For now we will continue to generate layer 61 in successive runs. So we need to delete this version of the new nwell layer (as well as the acute angle marks on layer 99) with the following commands:

**UNSELECT ALL
SELECT LAYER 61+99 ALL
DELETE**

Regenerate the layout file for the next DRC run with the DRC command. Exit the layout editor.

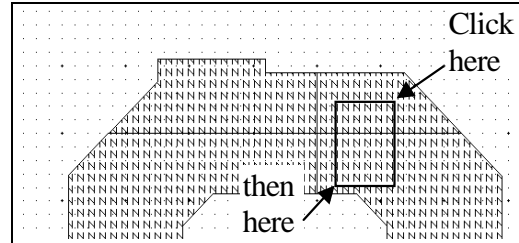


Figure 248: Selecting two polygons for merge.

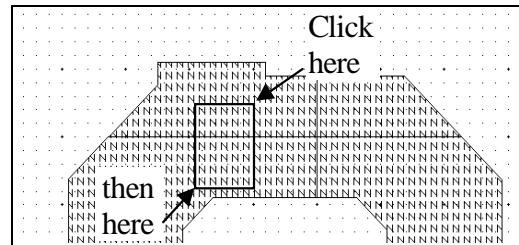


Figure 249: Selecting other two polygons for merge.

Be sure to remove the NO_WARN_ACUTE rule from the rule set used on your final design.

It is best to have mask layer generation rules in a rule set from an early stage to verify design rules with what will be the real mask layer. In your own designs you can choose whether to ignore the acute angle warnings until the design is complete, or to add the NO_WARN_ACUTE rule to the rule set used for preliminary checks so that no warnings need to be ignored. For this tutorial, add this rule now near the beginning of the rule set.

NO_WARN_ACUTE

Finding the Bad Polygon

Remember that the DRC console messages shown on page 423 warned us about a bad polygon. Bad polygons are a particular class of shapes with self-intersecting sides that can cause problems for mask processing software. By default, the DRC will warn you about all bad polygons on all defined input layers. Since bad polygons will never be created by DRC rules, only input layers are checked for bad polygons. (This is opposite the acute angle test that verifies output layers since acute angles can be created by the DRC on output layers.)

If you do not want bad polygons on unused layers to generate warnings, add the NO_CHECK_INPUT rule to the rule set.

Since all input layers are tested for bad polygons, why was this warning not present in earlier runs? The answer is that the pwell layer was not defined as an input layer in earlier DRC runs. Only layers that are defined as input layers in the rule set are checked for bad polygons. For this reason, **it is a good idea to define all mask layers in the layout as input layers in the rule set**, even if you have no design rules to test for some layers.

The warning about the bad polygon is expanded in the log file (DRCOUT.DLO in our case) with the following message:

```
1 bad polygons in cell ADVCKTP
  1 bad polygons on layer 2
    Bad polygon(s) or wire(s) appear on layer 99
of error file Q:\ICED\DRCTUTR\ADVCKTP.ERR in cell
ADVCKTP coordinates.
```

Since the bad polygons are found before the DRC flattens any data, only one warning is posted for a bad polygon in a subcell, even when that subcell is used many times. Since the error is found in a subcell, the error mark created on layer 99 is created in a command file created specifically for that subcell, ADVCKTP.ERR instead of the main command file DRCOUT.CMD. Other error shapes can be created in subcell error files as well; however, spacing violations are always marked in the main command file because they are not found until the layout data is flattened.

Open the layout editor now to edit the ADVCKTP cell (*not* the main cell ADVTUTR). Import the error mark created by the DRC by executing the subcell error command file with the following command:

@ADVCKTP.ERR

The error mark is a copy on layer 99 of the entire pwell shape. The misdigitized coordinates that make the pwell shape a bad polygon may not be obvious until you zoom in on the lower right hand corner with a VIEW BOX command.

This bad polygon was created by misdigitizing the final corner of a polygon. This can happen if you are digitizing the last corner of a large shape at a scale that is too large to show you fine detail. You try to close the polygon by redigitizing the starting vertex, but overshoot the vertex and digitize a point past the starting vertex. (See Point 1 in Figure 250.) If you simply click a few times in the vicinity you may digitize points similar to those in Figure 250 before redigitizing the starting vertex to close the polygon. You may not realize that you have not digitized the corner correctly.

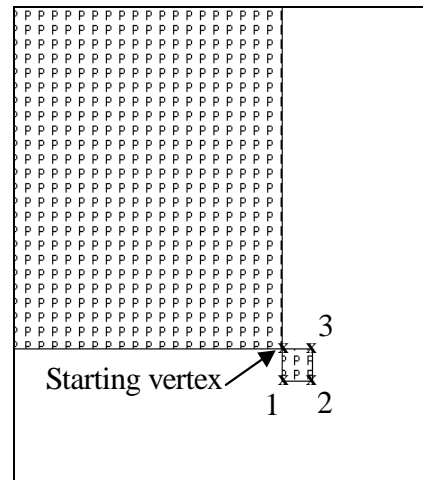


Figure 250: One way to accidentally create a bad polygon

When you digitize a polygon like this, the sides intersect each other as shown in Figure 251. This type of polygon definition is likely to cause problems for mask-processing software.

It can be a little tricky to edit a shape with intersecting sides. The **MOVE SIDE** command will sometimes fail when you try to uncross intersecting sides. You must select both sides of vertex 2 using the following commands:

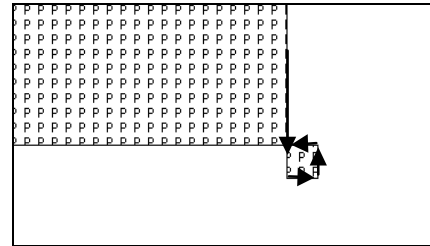


Figure 251: Self-intersecting sides of a bad polygon

UNSELECT ALL
SELECT LAYER PWELL SIDE IN

Then move the vertex to the starting vertex with the following command:

MOVE VERTEX

Once the vertex is moved successfully, delete the shape on layer 99 with the commands:

UNSELECT ALL
SELECT LAYER 99 ALL
DELETE

Save the cell and terminate the layout editor with the **EXIT** command.

Now we need to regenerate the layout data for the next DRC run. Open the layout editor to edit the ADVTUTR cell and use the **DRC** command to export the data. You can then terminate the editor.

Hierarchical Output

You should be familiar with the information on hierarchical processing beginning on page 134 if you want to use hierarchical output for real designs.

Subjects covered below
Dangerous processing options
Hierarchical output – generating shapes for import into nested cells
Replacing hierarchical output
Deleting hierarchical output

Except for the few errors found in subcells (e.g. bad polygons) the shapes generated by the DRC are usually created flat in one main cell. The nwell shapes that we created in the last exercise were all output at the main cell level, even though the nwell and active shapes used to create the new layer were nested in subcells.

This is usually the preferred method for creating new layers with the DRC. All shapes generated by the DRC are kept separate from the other cells in one cell at the main cell level where they can be deleted or replaced all at one time.

However, once you are comfortable with the DRC, you can use options to generate shapes hierarchically. Shapes created from shapes in subcells will be nested in separate subcells. These new subcells can be added automatically to each of the original subcells. This can save a significant amount of storage space. One new nwell shape stored in a subcell used 20,000 times in a chip takes much less disk space than 20,000 shapes in the main cell.

Of course, if you want to take your time, you can run the DRC flat on individual subcells one at a time and import the results into each subcell. However, hierarchical options let you perform this same procedure automatically on all cells in your design with a single DRC run.

In this part of the tutorial we will generate the new nwell layer hierarchically so that the nwell shapes remain in the subcells rather than being created at the main cell level.

Dangerous Processing Directives

Dangerous operations generate layers in subcells that may be invalidated by shapes in a higher level cell. The example used earlier in this manual is shown in Figure 252. Let us consider the following rule

$$c = a \text{ AND NOT } b$$

A shape on layer a is contained in a subcell and a shape on layer b is in a higher level cell. When you process layer c “dangerously”, the entire shape on layer a is copied to layer c while processing the subcell. When the DRC is processing the higher level cell it realizes that the section of the shape on layer c that is covered by the layer b shape was generated in error. Since there may be other copies of the subcell not covered by shapes on layer b, the DRC cannot solve the problem by going back and changing layer c in the subcell.

When layer c is processed “safely”, the DRC will not create shapes on layer c in the subcell. It will wait until it is processing the main cell and create the shapes correctly.

Throughout this tutorial we have used the ALL_SAFE rule in our rule set. This rule directs the DRC to process all dangerous operations in a safe manner. While this avoids problems like the one shown above, it results in many layers being generated at the main cell level rather than within subcells.

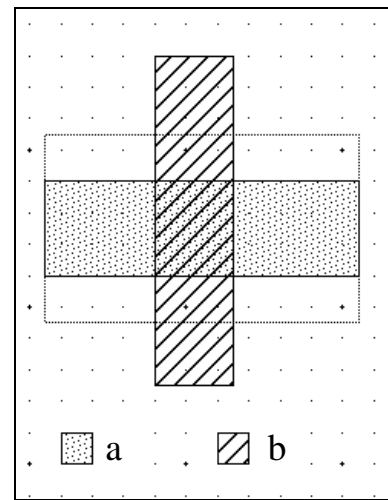


Figure 252: Layer A in subcell and layer B in main cell.

The DRC will post a warning message and generate error marks when dangerous processing causes errors.

See the list of dangerous operations on page 137.

Safe processing directives interfere with hierarchical processing since most or all shapes must be generated at the main cell level. Only layers that have no dangerous operations associated with them will be generated at the subcell level.

If we leave the ALL_SAFE directive unmodified, even when we use the hierarchical command line options all of the nwell shapes will still be generated at the main cell level. This is because the nwell generation rules include dangerous operations (i.e. BLOAT and AND NOT.)

If we used the ALL_DANGER rule instead of ALL_SAFE, we would have a different problem. Some of the scratch layers used for verification rules are generated by dangerous operations. The not_gate_overlaps layer in particular will have several processing errors when processed dangerously. The log messages would look as follows:

```
*****DANGER*****DANGER*****DANGER*****DANGER***
*****DANGER*****DANGER*****DANGER*****DANGER***
*****DANGER*****DANGER*****DANGER*****DANGER***
```

A logical error was made processing layer NOT_GATE_OVERLAPS in cell ADVTUTR. One of ADVTUTR's subcells contains a section of NOT_GATE_OVERLAPS that was removed by a logical operation in ADVTUTR. This means any further results in ADVTUTR or a cell containing ADVTUTR involving layer NOT_GATE_OVERLAPS are likely to be wrong.

An outline of the offending area (in cell ADVTUTR coordinates) appears on layer 99 of error file E:\ICED\DRC\ADVTUTR\DANGER\ADVTUTR.ERR. This outline can be used to locate the offending subcells.

4 layer NOT_GATE_OVERLAPS figures were outlined on layer 99.

What we need to do is restrict the dangerous processing to the generation of the nwell layer. We can do this with the DANGER_LAYER rule. We need to add the following rule to the rule set:

DANGER_LAYER nwell

Advanced Tutorial: Hierarchical Output

However, since the nwell layer is generated from the p_active, and p_active_bloat layers, they must also be generated dangerously. If those layers are generated safely, the shapes used to create the nwell shapes are already at the main cell level. So we must add the following lines as well:

```
DANGER_LAYER p_active  
DANGER_LAYER p_active_bloat
```

When adding the rules above to the rule set, place them after the layer definition rules. The danger properties of a layer cannot be set until after the layer is defined.

Command Line Options for Hierarchical Output

To get hierarchical output from the DRC we must add the HIERARCHICAL option to the DRC command line. The syntax of the option for our testcase is as follows:

When typing this option in a Windows shortcut, replace the '=' with a '#' to avoid misinterpretation of the command line.

```
HIER=_WELL
```

The HIERARCHICAL keyword has been abbreviated here to “HIER”. The string “_WELL” will be added to the end of every cell name in the hierarchical output data. The nwell for cell ADVCKT will be created in a cell with the name ADVCKT_WELL. We will demonstrate how this cell is added to the ADVCKT cell after we run the DRC.

Other command line options should be combined with the HIERARCHICAL option.

The NO_HIER_WARNING directive can be specified in the rule set or on the command line.

Since the DRC is processing some layers safely despite the HIERARHICAL option, the DRC will warn you that some of the output data will not be hierarchical. You would need to reply to this warning prompt with a keystroke to continue with the DRC run. To avoid this warning prompt, we need to add the following option to the command line.

NO_HIER_WARNING

The DRC will automatically flatten some cells during preprocessing. Subcells containing 5 or fewer shapes will be flattened by default. So will subcells used only once. To prevent this we need to add the following option to the command line:

NO_FLATTEN

Creating and Importing the Hierarchical Data

Recompile the rule set. Then execute the DRC. The entire DRC command line should now look like the following:

```
DRC3-NT25 ADVTUTR ADVTUTR DRCOUT SLOW ...  
... HIER=_WELL NO_HIER_WARNING NO_FLATTEN
```

The run should end with messages similar to the following

```
9 total figures output to non-error layers.  
0 total figures output to error layers.  
*****DANGER*****DANGER*****DANGER*****DANGER***  
*****DANGER*****DANGER*****DANGER*****DANGER***  
*****DANGER*****DANGER*****DANGER*****DANGER***  
This run may have incorrect answers --- Read your log  
file Q:\ICED\ADVTUTR\DRCOUT.DLO
```

This how the DRC alerts you to errors in subcells caused by dangerous operations. We will look at the log file and diagnose the problem in a moment.

²⁵ The executable file name for released versions for Windows is DRC3-NT.EXE.
The executable file name for beta Windows versions is DRC3-NTX.EXE.

First we need to create a temporary cell from which we will import the results of the DRC run. We need a temporary cell since the command file that will add the newly created cells to the existing design cells must be executed while none of the design cells are open.

When we perform the import operation in the temporary cell, the procedure will modify all of the design cells to add the newly created cells. The temporary cell's environment settings (e.g. layer names, color properties, grid definitions) will be saved in all of these modified cells. If the temporary cell has different environment settings than the original design cells, the old environment settings would be lost.

So, we need to create this temporary cell with the same layer and other environment settings as our design cells. If all of your environment settings are defined in a startup command file so that all new cells have the same properties, then any temporary cell you create will have the appropriate environment automatically. However, we have added unique layer properties in our main cell for DRC generated layers. (In the case of real designs, the main cell may be modified with many layer definitions for DRC generated layers that may include layer names, patterns, and colors.) We don't want to lose these layer definitions when the environment settings get replaced.

The
TEMPLATE
command in the
layout editor
can be used to
export all
environment
settings into a
command file.

There are several ways to get the environment of our main cell stored in the new temporary cell, but the easiest way is to create the temporary cell after the layout editor has been opened to edit the main cell. Open the layout editor now to edit the ADVTUTR cell. Now execute the following commands:

```
EDIT CELL TEMP  
EXIT  
QUIT
```

Now open the layout editor to edit the cell we have just created with the name "TEMP". From this cell we can execute the command files created by the DRC. First execute the main command file with following editor command:

```
@DRCOUT.CMD
```

This command file will create all of the new cells generated by the DRC. All cell names will end with “_WELL” as we specified in the command line. The existing design cells are still unmodified at this point.

Now we need to execute an additional command file generated by the DRC to actually add the new cells to your design cells. These procedures are kept separate because if you repeat this process, you do not want to add additional copies of the new cells to your design cells. (We will cover how to replace the hierarchical DRC output with new results on page 438.) Execute this additional command file with the following layout editor command:

@DRCOUT.ADD

Now the new cells are added to each of the design cells. Now exit the editor to save all of the cell files.

Fixing a Dangerous Processing Error

Now that we can look at the results, let us diagnose the error reported in the log file. The warning in the log file will look similar to the following:

```
*****DANGER*****DANGER*****DANGER*****DANGER***
*****DANGER*****DANGER*****DANGER*****DANGER***
*****DANGER*****DANGER*****DANGER*****DANGER***
A logical error was made processing layer NWELL[61] in
cell ADVTUTR. One of ADVTUTR's subcells contains a
section of NWELL[61] that was removed by a logical
operation in ADVTUTR. This means any further results in
ADVTUTR or a cell containing ADVTUTR involving layer
NWELL[61] are likely to be wrong.
    The problem can be corrected by specifying that
    layer NWELL[61] or the problem subcells (not ADVTUTR) be
    ungrouped.
    An outline of the offending area (in cell ADVTUTR
    coordinates) appears on layer 99 of error file
    E:\ICED\DRC\ADVTUTR\DANGER\ADVTUTR.ERR. This outline
    can be used to locate the offending subcells.
```

Let us look at the error mark generated by the DRC. Open the layout editor to edit the ADVTUTR cell.

Now we need to import the file indicated in the error log message, ADVTUTR.ERR. This is a different file than the main DRC command file. Dangerous processing error marks are stored in a subcell error command file. It just happens that in our case the cell where the error is found is the main cell, so this subcell error file is named for the main cell. Execute the following commands in the editor.

```
@ADVTUTR.ERR  
UNSEL ALL  
SEL LAY 99 ALL  
VIEW SEL  
VIEW OUT 4
```

At this scale, it is relatively easy to see that the mark on layer 99 (a line component in this case) is marking the area where the pwell shape overlaps the new nwell shape. (See Figure 253.) The new nwell shape on the left was created nested inside the cell ADVCKT, and the pwell shape on the right is nested in the cell ADVCKTP. When the DRC processed the main cell and flattened both cells to test for dangerous processing errors, the rule “nwell = nwell AND NOT pwell_in” rule could not be performed correctly for this area.

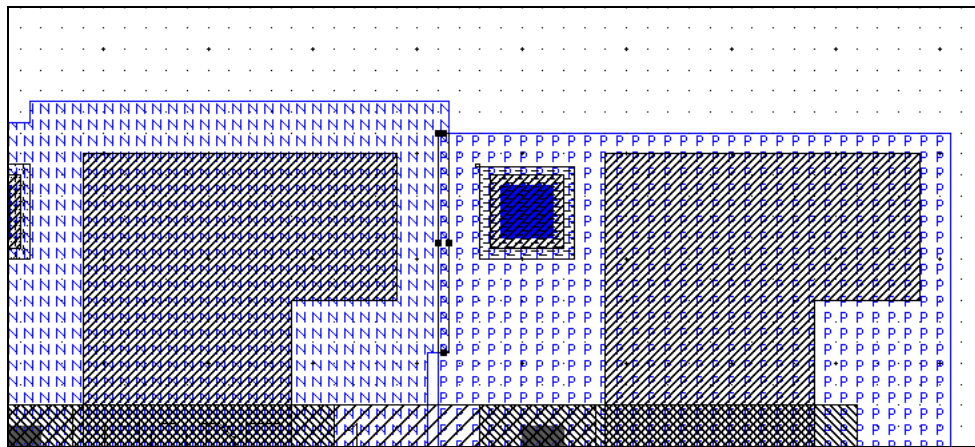


Figure 253: Dangerous processing error mark selected

You have options on how to solve errors like the one above. The easiest method is to move the ADVCKTP cell slightly to the right. However let us assume that space is at a premium and that it would be better to shift the side of the active shape to the left to fix the problem. Now when the active shape is bloated, it will not cause the new nwell shape to overlap the pwell shape. This will require the recreation of the nwell layer.

We need to delete the error mark on layer 99 and shift the side of the active shape in cell ADVCKT with the following commands:

```
UNSEL ALL
SEL LAY 99 ALL
DELETE
PEDIT NEAR
SEL SIDE NEAR
MOVE -1,0
EXIT
```

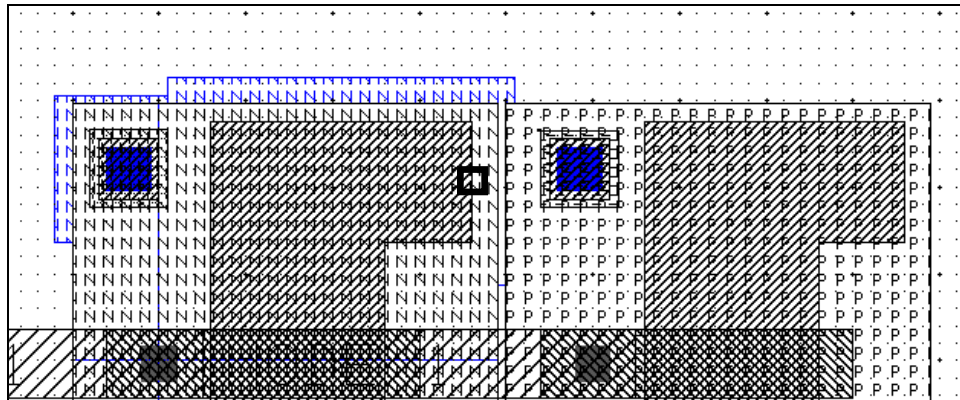


Figure 254: Correct select box for selecting cell ADVCKT and active side

For the “PEDIT NEAR” and “SEL SIDE NEAR” commands, use the cursor to position the select box as shown in Figure 254. Note that all copies of the ADVCKT cell have been changed.

Now recreate the DRC data with the DRC command and rerun the DRC with the same command line shown on page 433. If any other error messages are posted, fix the problems before continuing.

Replacing Hierarchical Output

The command files created by the DRC now contain the corrected nwell layer on layer 61. However, if we execute the DRCOUT.CMD command file now to add these new shapes to each _WELL cell, both the old shapes and the new shapes will be contained in each cell.

If the cell structure has not changed, we can recreate all of the _WELL cells with DRCOUT.CMD, but leave the cell references nested inside all of the original cells. The DRCOUT.ADD command file does not need to be executed.

Close the layout editor. Using your favorite method, delete all of the _WELL cell files. In the console window, use the following DOS command:

```
DEL *_WELL.CEL
```

Open the layout editor to edit the temporary cell, TEMP.CEL. Execute DRCOUT.CMD command file and exit the editor with the following commands:

```
@DRCOUT  
EXIT
```

This recreates the _WELL cells. The cell files are now saved to disk.

Do not re-execute DRCOUT.ADD since the _WELL cell references are still included in the original cells.

Deleting Hierarchical Output

If you want to delete all of the hierarchical results from the DRC, copy the DRCOUT.ADD command file to DRCOUT.DEL. Edit the DRCOUT.DEL file. For each of the “ADD CELL=“xxx_WELL” AT (0.0, 0.0)” commands, replace the command with the following set of commands:

```
UNSEL ALL  
SEL CELL xxx_WELL ALL  
DELETE
```

Where “xxx_WELL” should be replaced with each appropriate cell name.

The temporary cell should be created with the environment of a design cell. See page 434.

Execute the DRCOUT.DEL command file while editing the temporary cell. This will remove the hierarchical results from each of the original cells.

Speeding Long DRC Runs

Subjects covered below
Separating long reach and short reach rules into rule subsets
Pad size verification using MIN_WIDTH rule
Optimizing panel size

The most important methods to improve DRC efficiency are to optimize the panel size and panel border.

The DRC can process only small designs as a single panel. Larger designs are divided into panels and processed one panel at a time. Only one panel is flattened at a time, allowing the rest of the design to remain hierarchically nested which saves large amounts of storage space. This panel processing allows the verification of entire chips with only the resources of a typical PC. However, the overhead of panel processing increases run time, especially when the panel size is not optimized for a specific design density on a PC with specific storage capacity.

Add the
SHOW-
_BORDER
option to the
DRC command
line to see how
the panel border
is calculated by
the DRC.

When you slice a design into panels for verification, spacing rules must look beyond the boundary of a panel to verify that no shape just outside the boundary is too close. In order for shapes near or crossing a panel boundary to be processed correctly, the DRC must include a border around all sides of each panel. The panel border is automatically calculated by the DRC based on the layer with the maximum reach as determined by the rules. **Reach** is defined as the minimum border distance that insures that no violations will be missed or marked as false errors.

You may have noticed that the DRC took longer to execute once we began using a panel size of 50x50. A large part of this delay is caused by the metal2 spacing rule that requires a border of 30 microns on each side of each panel. This wide

border forces most shapes to be tested many times due to the overlapping borders on the panels.

Our artificially small panel size magnifies this problem, but all long reach rules will cause this type of extra processing when the design is divided into panels.

The extra time involved to process shapes in a large border is multiplied by each rule processed in the same pass, even those rules with a relatively short reach. The same panel border is used by most spacing rules.

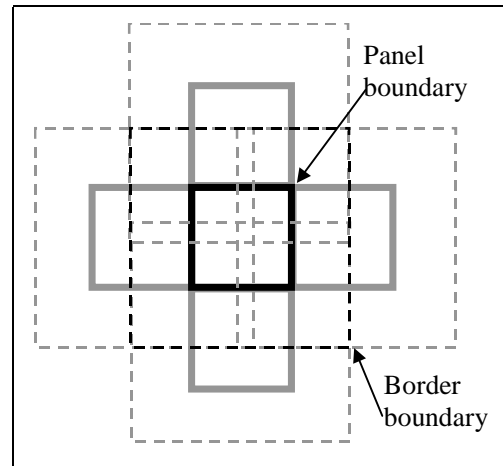


Figure 255: Neighboring panels with overlapping long reach borders

The DRC would be far more efficient if only the long reach rules were processed with the large panel border. If the shorter reach rules were processed with a small border, far less processing time would be needed.

Testing Minimum Pad Size

Let us add another long reach rule to our rules file to highlight the problem of large panel borders. Let us assume that the minimum dimension of glass shapes in pad constructs is 50 μ m x 50 μ m. We need to verify this with a MIN_WIDTH rule. Open the rule set for editing now. Add the following rule after the m2_pad_spacing_err rule:

small_glass = MIN_WIDTH(glass, 50)

We need to define the new error layer even though our design will have no violations of this rule. Add the following layer definition rule with the others:

OUTPUT ERROR LAYER 109 small_glass

Separating Long Reach/Short Reach Rules into Rule Subsets

When you use rule subsets to separate long reach rules from short reach rules, you can speed the DRC considerably. You can use the log files to find which rules are causing a large border. Save the rules file now and compile it. Then execute the DRC with the SHOW_BORDER command line option. The DRC command line should now be as follows:

DRC3-NT²⁶ ADVTUTR ADVTUTR DRCOUT SLOW SHOW_BORDER

The border calculations are shown in the log file (DRCOUT.DLO in our case.) Open this file now to see the border calculations. The following lines should be near the beginning of the file.

```
***Deriving border for pass 3
19. SMALL_GLASS[109]= MIN_WIDTH(GLASS[16], 50/~DET)
reach(SMALL_GLASS[109])=50.
:
:
***Deriving border for pass 7
34. M2_PAD_SPACING_ERR[104] = MIN_SPACING(
    METAL2,
    PAD_METAL2,
    30
    /~CONN/P/OVER/CROSS/T/END/~DET)
reach(M2_PAD_SPACING_ERR[104])=30.
```

The bolded lines above show us that the rule that creates the small_glass layer and the rule that creates the m2_pad_spacing_err layer are the rules causing the longest reaches.

²⁶ The executable file name for released versions for Windows is DRC3-NT.EXE. The executable file name for beta Windows versions is DRC3-NTX.EXE.

See another example of testing long reach rules efficiently on page 164.

If you could test only the long reach rules with the large border, and test all short reach rules with a smaller border, the short reach rules could be processed much faster. We can do this using rule sets without dividing the rules into two files. The same rules file can be used to test different sets of rules without further modification of the file. We specify on the DRC command line which rule subsets should be executed.

The RULE_SET rule is used to identify subsets of rules that can be executed by the DRC instead of executing all of the rules in a file. Add the following rule near the top of the rules file:

RULE_SET LONG_REACH SHORT_REACH

Now we bracket the verification rules into named subsets. **It is important to bracket only the verification rules.** Layer processing rules and/or connection rules that are unnecessary are removed by the DRC automatically. However the DRC cannot remove unnecessary rules if they are specified in a rule subset. The DRC may crash quickly with a message similar to the following if unnecessary layer generation or connection rules are contained in the specified rule subset.

```
You selected rule:
  30. CONNECT METAL2 PAD_METAL2
      In set SHORT_REACH
      which generates a connection never used.
Run canceled.

***CRASH*****CRASH*****CRASH*****CRASH*****
```

First we bracket only the two rules that result in a long reach. (The unbolded lines below should already be present.)

```
LONG_REACH ON
m2_pad_spacing_err = MINSPACING(metal2, pad_metal2, 30 /~CONN)
small_glass = MIN_WIDTH(glass, 50)
LONG_REACH OFF
```

Now we bracket the rest of the verification rules in the other rule subset.

SHORT_REACH ON

```
nwell_sp_err = MINSPACING(nwell, nwell, 10 /~CONN)
gate_cpoly_sp_err = MIN_SPACING (gate, cpoly, 2)
poly_unconn_contact_err = MIN_SPACING(poly, cpoly, 4 /~CONN)
gate_overlap_err = MIN_SPACING (gate/OUT, poly/IN, 2)
```

```
gate_overlaps = poly OR active
not_gate_overlaps = NOT gate_overlaps
gate_no_overlap_err = gate TOUCHING not_gate_overlaps
```

SHORT_REACH OFF

To specify that the DRC should verify only the long reach rules, execute the following command line (the parentheses around the name of the rule subset are required):

When typing this command line in a Windows shortcut, replace the '=' with a '#' to avoid misinterpretation of the command line.

```
DRC3-NT ADVTUTR ADVTUTR DRCOUT SLOW ...  
... DO=(LONG_REACH)
```

The rules in the SHORT_REACH rule subset are not executed; neither are the layer generation rules that are not necessary to execute the LONG_REACH rules. If you look at the DRCOUT.DLO log file, you can see that the unnecessary rules have not been executed, including the generation of the gate and not_gate_overlaps layers. This DRC run should have taken less time than the execution of the entire rule set. You can then execute only the short reach rules by specifying the SHORT_REACH rule subset in the command line.

When you divide the rules this way, you can probably run the long reach rule subset less often. The long reach rules are typically important mainly in areas of the chip that do not change that often, e.g. pad constructs.

Using Rule Subsets for Very Long DRC Runs

You do not have to separate long reach rules from short reach rules to use rule subsets. You can arbitrarily separate the verification rules into as many subsets as desired. Any DRC run that executes only a subset of the rules will execute more quickly.

Let us assume that you are verifying a large design with a long rule set. Divide the rules into two subsets and execute the DRC on the first subset with the appropriate DO option. Then after the first run is finished, run the DRC again on the second subset. The total execution time of the two DRC runs will be longer than if you ran the DRC on the entire rule set, however, you will have results for the first subset more quickly. Then you can fix the errors found by the first subset while the DRC is executing the second subset.

When you want to execute all the rules in a single run, just leave the DO option off of the DRC command line. You do not need to remove the rule subset lines from the rules file.

Optimizing Panel Size

Panel size is directly related to the amount of memory the DRC requires. A large design must be divided into panels so that only a small portion of the layout is flattened and in memory at any given time.

However, panel size also has an important impact on how quickly the DRC will execute. Extra processing is involved, especially in testing areas where panel borders overlap.

Since there is a trade off between extra processing required for panel processing and storage saved due the smaller amount of data stored in flattened form at any given time, time may be saved by increasing or decreasing the number of panels.

This lesson will be quite a bit more time consuming than our other exercises have been. Your computer will be tied up for some time. You may want to spread this lesson over a few lunchtimes.

Our current testcase is too small to adequately explore panel size trade off. If you have a large design that you can use as your testcase, you may want to use that layout instead of the input file we build next. The error status of the layout is not important. We are not fixing errors in this lesson, only optimizing run time by modifying the panel size.

Advanced Tutorial: Speeding Long DRC Runs

If you have no layout file suitable for a test of a larger design, expand the size of the ADVTUTR layout. Open the layout editor to edit the ADVTUTR cell. Add an array of cells to add a large amount of layout data with a single command. Type the following command:

ADD ARRAY=ADVCKT N=(100,100)

Once you place the corner of the array with the cursor, you have added 10,000 cells to the design. Now create the file for the DRC and exit with the commands:

**DRC
EXIT**

You can use the same rules file we used in the last lesson, or you can modify it to use MINSPACING rules that verify similar distances to the rules you will be using on your real data. If you edit the file, save and compile it.

You can add these options to the DRC command line or in the rules file.

You should test various panel sizes in the rules file. First use the existing panel size settings added on page 420.

**PANELX =50
PANELY =50**

This is most likely to small a panel size for a realistic design, but it provides a suitable minimum panel size. Now run the DRC and look at the run time statistics at the end of the log file, DRCOUT.DLO. The listing should look similar to the lines below:

```
Scratch file size=0 bytes.
Information was written to the scratch file 0 times.
Information was read from the scratch file 0 times.
Running time:
3,967,256 total timer on/offfs
  Total   1:11:03 (1 times)
  Disc swaps      0
```


The listing above indicates that no disk swapping to a scratch file was necessary for this run. If any of your runs indicate that the scratch file was used, try smaller panel sizes if possible. Disk swaps add to processing time considerably.

The run time for the run above was 1 hour and 11 minutes. Note the time of your run and the panel size used.

For the next run, try removing the PANEL rules entirely. This lets the DRC attempt to find an optimal panel size. Try different panel sizes in other runs until you have optimized run time. Be sure to save and compile the rules file each time you change the panel size.

On a sample computer, we ran the DRC on the entire rule set. We limited the DRC to using 32Megabytes of memory with the “HOG=32” command line option. See the results of various panel sizes in Figure 256. The run with no panel size restriction did not end successfully. It was cancelled with the <ESC> key in less than an hour since little progress was being made. The log file looked similar to the thrashing example shown on page 120.

You can see the dramatic effect of optimizing panel size.

You can cancel any run early by pressing the <ESC> key if desired. The log file is still created, and you can get a good feel for how long the entire run will take by watching the progress of the first pass with the console messages.

Panel size	Run Time in minutes
50x50	71
150x150	10
250x250	7
500x500	10
No PANELx rules- default calculated to 1428x1392	>60

Figure 256: Various run times for various panel sizes

Conclusion

This concludes the tutorial. We have covered all of the tasks common to most large scale designs.

Continue to experiment with these files to test aspects of the DRC as needed. Edit the testcase rules file to test design rules more unique to your designs. Refer to the reference sections of this manual to learn more about various additional rules and aspects of the program needed for more advanced testing.

If you run into problems, please consult the trouble-shooting guide on page 27 before contacting technical support.

Good Luck and Enjoy.

Appendix A: Obsolete Syntax

Obsolete DRC Rules

If you want to use these obsolete rules only for old versions of the DRC, see the version control rules 2_ONLY and 286_ONLY.

The following rules were developed for previous versions of the DRC. These rules are still supported by the current version of the DRC to enable users of old rule sets to use the current version of the program without forcing them to modify their rule sets.

The correct rule in the current version to use in place of each obsolete rule is mentioned at the beginning of each description.

MAX_QUAD

Limited air bridge recognition

error_layer = **MAX_QUAD** (*layer1*, *max_length*)

This rule has been replaced by the more thorough air bridge recognition rule BRIDGE described beginning on page 196. The main problem with the MAX_QUAD rule is that missing posts at the end of air bridges will not be marked as errors.

The MAX_QUAD rule will mark as errors all shapes on *layer1* that meet either of the following conditions:

the shape does not have exactly 4 sides

or

at least one side of the shape is longer than *max_length*

To use this rule you will need to use the BLOAT rule to expand the layer that represents the posts so that the expanded post shapes overlap the metal layer of the bridges. Next, you need to subtract the expanded post layer from the metal layer with a rule similar to:

BRIDGE = METAL AND NOT BLOATED_POST

The MAX_QUAD rule should be written to verify the new BRIDGE layer. Bridge corners without posts should be found. However, bridges with missing posts at the ends will not be found. Slanted bridges may or may not be classified correctly.

Be sure to subtract twice the value used to bloat the post layer from the maximum bridge length when specifying *max_length* in the rule.

RECTANGLES*Find shapes that are not rectangles of specific sizes*

error_layer = **RECTANGLES** (*layer1*, *size1* [, *size2* [... , *sizeN*]])

This rule has been replaced by the more robust IS_BOX rule. (See page 222.) One main difference between the two rules is that the IS_BOX rule does not automatically classify the shapes it creates as errors.

The RECTANGLES rule will consider all rectangles that fit the size criteria as valid shapes. All other shapes on *layer1* will be copied to *error_layer* and counted as errors. To pass the test, rectangles must be square with the axes (i.e. the sides must be vertical and horizontal).

(Remember that all shapes on the same layer are merged by the DRC. Rectangles that touch another shape on the same layer will be merged during preprocessing. When a rectangle is merged with touching shapes, the resulting shape may no longer be rectangular.)

You must specify at least one *sizeN* parameter. You may specify up to ten different *sizeN* parameters. You can type additional *sizeN* parameters on different lines for readability, but you cannot start a new line in the middle of a *sizeN* parameter.

The syntax of each *sizeN* parameter is:

(*xmin* [: *xmax*], *ymin* [: *ymax*])

To allow the dimensions of the rectangles to be in a range, specify both the minimum and maximum dimension separated by a colon (:). To specify an exact dimension, type only the minimum value. When the maximum value is not included, it is assumed to be equal to the minimum. Each dimension must be a positive real number. The units are the user units in the ICED™ cell.

The syntax of the *sizeN* parameters is exactly the same as the syntax used in the IS_BOX rule. See that rule for many examples of *sizeN* parameters.

SKIPPED_POLY *Assign layer number for shapes unknown to DRC*

SKIPPED_POLY = *layer_number*

In the past, new versions of the ICED™ layout editor have supported shapes that the DRC was not able to process correctly. This version of the DRC supports all shapes that can be created by the current versions of the layout editor. However, if the layout editor is enhanced before you receive a new version of the DRC, or if you are using an older version of the DRC, the DRC will classify shapes it cannot process correctly as "skipped" and copy them to layer number 99 by default.

You can add the SKIPPED_POLY rule to change the layer number where the DRC will store skipped shapes.

OUTPUT LAYER Obsolete Keywords

The following keywords of the OUTPUT LAYER rule are considered obsolete. They all have exact correspondences to syntax documented in the current syntax of the OUTPUT LAYER rule covered beginning on page 284.

MASK

The MASK keyword is exactly equivalent to the POLYGON keyword. It indicates to the DRC that the layers defined with the rule should contain only polygon shapes suitable for mask generation as opposed to error wires. If you attempt to use a layer defined with this keyword as the error_layer for any of the rules that generate wires (as shown in the table on page 62), you will receive an error message from the rules compiler.

OUTLINE

The OUTLINE keyword is exactly equivalent to the WIRES keyword. Shapes on layers defined with an OUTPUT OUTLINE LAYER rule will be converted to wires that outline the edges of the shapes during DRC output. This is useful primarily to allow all of your error layers to have similar properties.

OUTPUT ERRORS

This syntax is exactly equivalent to:

OUTPUT WIRE LAYER

OUTPUT GEOMETRY

This syntax is exactly equivalent to:

OUTPUT POLYGON LAYER

Index

- ! Comment character.....153, 173
- #
 - used to replace =333
- \$D3RVIRT.000323, 324
- \$D3VIRT.000.....332, 341, 362
- % variable layer number indicator58
- % variable layer number indicator346
- %n batch file parameters359
- & continuation character.....172, 221
- () not allowed in Boolean layer rules.....64
- * wildcard in cell name specifications208, 219, 297
- ... used to indicate continuation175
- ; delimiter218
- ; Semicolon characters.....172
- @*file_name* layout editor command.....368, 369
- @MAIN cell name specification219
- @*opt_file* command line option.....336
- [] used to indicate optional keywords174
- _ underscore character172, 333
- { } used to allow rules to span several lines218
- | delimiter in cell name specifications219, 297
- + used to combine input layers218
- =
 - disappearing from command line333
- = use in assignment rule.....187, 281
- 2_ONLY rule.....**176**
- 286_ONLY rule.....**178**
- 3_ONLY rule.....**179**
- Acute angles
 - bloats126
 - finding and fixing423
 - finding notches on specific layer231
 - finding on all output layers.....313
 - finding protrusions on specific layer242
 - in bloated shapes191
 - in mask layers.....76
- in MIN_ANGLE and MAX_ANGLE rules107
- in MIN_NOTCH rule 106
- in MIN_WIDTH rule 104
- listed in log file.....364
- output layer number.....313
- suppressing check.....280, 314
- ADD file extension 147, 374
- ADD layout editor command
 - example373
 - used in DRC command file366
- Adjacent sides
 - verifying angles 107
- Advanced
 - Tutorial.....379
- ADVTUTR.CEL.....382
- Air bridge recognition.....196
 - obsolete version.....450
- Algorithm Options337
- ALL_SAFE rule
 - use in advanced tutorial384
- ALL_DANGER rule.....**180**
 - importance in hierarchical output148
 - overriding for specific layer299
 - use in hierarchial output277
- ALL_SAFE rule.....**181**
 - overriding for specific layer209
 - overview141
 - problems in hierarchical output431
- ALLOW_QUICK command line option.....338
- ALLOW_QUICK rule**182**
- AND NOT rule
 - not sufficient to verify enclosure312
- AND rule**183**
 - example in advanced tutorial.....383
 - simple example.....64
 - using to find overlap errors85

Index

- Angled sides *See also* Acute angles
 - Exceptions to MIN_SPACING violations 93, 256
 - in output shapes 132
- Angles
 - overview of MIN/MAX_ANGLE 107
- Area
 - classify shapes by area 66
 - classifying hole coverage 211
 - design area reported in log file 363
 - finding shapes less than minimum area 243
 - limiting design area checked 350
 - overview of MIN_AREA rule 107
 - overview of MIN_FILL rule 109
 - restricting area checked 159
 - verifying layer coverage 245
- ASPECT_RATIO rule **184**
 - reach 125
- Assignment rule **187**
 - 319
- Automating DRC import in the layout editor.. 370
- AWAY option of MIN_SPACING rule .. 93, 256
- Backups
 - comparing two designs 335
 - of previous DRC run 361
- Bad polygons
 - coordinates 376
 - defined 74
 - finding and fixing 426
 - importing error shapes 375
 - listed in log file 363
 - output layer number 189
 - restricting check to used layers 276
- BAD_POLY rule **189**
 - remove BADPOLY=0 in final run 169
- Base layer in bipolar transistors 114, 115
- BAT file extension 359
- Batch file 359, 385
 - PAUSE command no longer required 322, 345
- Batch file
 - avoiding user interaction 182, 279
- BB file extension 320, 325
- Bent sides 79
- Beta test 95, 258, 331
- Binary layout data file 16, 47, 334, 382
- Bipolar transistors
 - sample layer processing 114
- BLANK layout editor command 20, 371
- Blank spaces 172
- BLINK layout editor command 371, 372
- Bloat angle
 - reported in log 327
- BLOAT rule 68, 164, **190**, 419
 - reach calculations 126
 - using to classify wires 65
- BLOAT_ANGLE rule 68, **191**
 - reach calculations 126
- Bloats of acute angles 126, 191
- Boolean operations 63
 - adding to MIN_SPACING tests 85
 - AND 183
 - effects of Hierarchical processing 136
 - example of counting results as errors 85
 - NOT 187, 281
 - not sufficient to verify enclosure 312
 - OR 283
 - TOUCHING rule more robust for verification 68
 - XOR 316
- Border *See also* Panel border
 - rewriting rules to reduce 163, 441
 - testing enclosure 90
- BORDER command line option 348
 - caution when using 127
- BORDER rule **193**
- Borders of panels 124, 348, 440
 - overriding border on command line 348
- BOTTOM command line option 159, 351
- Boundary of design area 350
 - reported in log file 363
- Bounding box
 - of entire design 245
 - of shapes 64
 - storing cell boundary as shape 221
 - using aspect ratio to classify shapes 184
 - using cell boundary to classify shapes 221

-
- using size to classify polygons194
 - Bounding box of a cell.....60
 - BOUNDS rule**194**
 - reach calculations126
 - Bow tie shapes
 - locating.....74
 - Boxes222
 - BRIDGE rule**196**
 - adding tolerances to log file350
 - ignored if QUICK_PASS used.....131, 337
 - Bulk layer
 - verifying poor conductors116
 - BY keyword in STAMP rule308
 - BY keyword of CONNECT rule201
 - CAP option of MIN_SPACING rule259
 - CAP=*angle* in MINSPACING rule95
 - Case
 - in DRC rules.....172
 - Cell boundaries
 - using to classify layers.....221
 - Cell flattening*See also* Hierarchy
 - preventing in input352
 - Cell hierarchy*See also* Hierarchy
 - preserving in input.....352
 - Cell Hierarchy Options.....352
 - Cell names
 - in hierarchical output.....147, 354
 - recorded in log file364
 - using to classify layers.....215, 219
 - Cell ungrouping*See also* Hierarchy
 - preventing in input352
 - Cells
 - bounding box.....60
 - classifying shapes by cell59
 - combining with SECOND_CELL option54
 - creating hierarchical output.....354
 - definition40
 - flattening automatic for cells used once353
 - flattening done automatically for small cells.....353
 - flattening on input144
 - how hierarchical data is generated134
 - isolating DRC shapes from original data.....369
 - overview of hierarchical processing 134
 - saving DRC data in separate cell..... 356
 - turning off display of design cell..... 371
 - CFLATTEN command line option 145, 353
 - Checklist for Final Run..... 168
 - Circles..... 64
 - using to classify polygons..... 225
 - Classifying Shapes by Distance 67
 - Classifying Shapes by Size or Shape 64
 - Coincident edges..... 93
 - avoiding marking as MINSPACING errors.....391
 - finding 86
 - finding touching shapes 397
 - in MIN_SPACING rule..... 90
 - Collector layer in bipolar transistors..... 114
 - Combine the data in two cells 54
 - Command file *See* DRC command file
 - conserving disk space 134
 - generic overview 52
 - importing DRC results..... 387
 - Command File Options..... 355
 - Command line
 - DRC 329
 - DRC rules compiler..... 319
 - options file..... 336
 - Comments
 - example in advanced tutorial..... 383
 - in DRC rules file..... 173
 - in options file..... 336
 - in rule sets 153
 - Comparing layouts..... 54, 335
 - Compiler*See also* DRC rules compiler
 - Compiler *See also* D3RUL-NT.EXE
 - Compiler *See also* Rules compiler
 - Compiler log file..... 325
 - Compiling DRC rules file 319
 - Conductive layers 110, 408
 - finding opens through poor conductors 116
 - removing material from 113
 - CONN option of MINSPACING rule..... 405
 - CONNECT rule **200**
 - added to process touching shapes..... 129
-

Index

- always processed safely.....139
- causes multiple passes129
- errors due to design area restrictions350
- examples in advanced tutorial404
- ignored if QUICK_PASS used.....131, 337
- importance of not using for poor conductors116
- listing.....327
- MIN_SPACING example267
- overview.....110
- problems connecting to wells411
- Connected layers.....410
- Connection groups
 - diagnosing problems413
- Connections
 - finding opens through poor conductors.....116
 - in MIN_SPACING rules99
- Console messages343, 386
 - progress reports increase run time.....165
- Console window11
 - closing25, 322, 345
- CONST rule.....153, 174, **203**
 - example in quick tutorial.....14
 - report326
- Constants153
 - in DRC rules.....203
 - including by file reference.....216
 - listing of in compiler log326
- Contact layers201, 408
- Contacts
 - eliminating false errors.....36
 - in bipolar processing114
 - in CONNECT rule.....111
 - overview of enclosure verification37
 - verifying coverage.....87
- Continuation lines.....172
- Coordinates
 - design boundaries listed in log file.....363
 - in log file362
 - listed by detailed logging50
 - not normally listed in log file49
 - of bad polygons.....376
 - of errors in command file366
 - of errors in subcell command file 373
 - shifted due to grid problems 79
- Copy protection 10, 28
- Copying a layer..... 187
- Copying files
 - sample DOS commands..... 13, 381
- Corners
 - treating differently in spacing check..... 96
- Coverage
 - verifying layer coverage 245
- Crashes 27
 - insufficient memory..... 162
 - try PANEL_VERTICES rule 123, 291
 - try smaller panels..... 118
- Crossing shapes
 - in MIN_SPACING rule..... 85
- Crossing sides 97
 - handling specially in spacing check..... 260
- Ctrl key 324, 332
- Current directory
 - sample of DOS command to change.... 13, 381
- Current drive
 - sample of DOS command to change..... 13
- CUT layout editor command 77
- CUT_RESOLUTION rule **205**
 - effect on inverse of layer 188
 - number listed in log file..... 363
- D3RULES.EXE
 - other executable names..... 319
- D3RUL-NT.EXE 318
 - command line 319
 - command line in advanced tutorial 385
 - command line in quick tutorial 14
- Danger errors 431
 - fixing 435
- DANGER logical error message..... 140
- DANGER_CELL rule..... **207**
 - overriding for specific layer 299
- DANGER_LAYER rule **209**
- Dangerous operations 136
 - ALL_DANGER rule 180
 - ALL_SAFE rule 181

-
- DANGER_CELL rule207
 - DANGER_LAYER rule.....209
 - methods of avoiding.....141
 - SAFE_CELL rule.....297
 - SAFE_LAYER rule.....299
 - Dangerous processing
 - cells used once flattened automatically353
 - creating hierarchical output.....354
 - importing error shapes.....376
 - preventing by flattening input data.....352
 - small cells flattened automatically.....353
 - Dangerous processing options429
 - Database *See also* Binary layout data file
 - reserving memory for162
 - Date stamp
 - cause of DRC warning394
 - listed in DRC log file363
 - Default panel size120
 - DELDRC.CMD368
 - Deleting DRC shapes.....389
 - Deleting results of hierarchical output147, 439
 - Design area
 - options in DRC command line350
 - reported in log file.....363
 - restricting area checked.....159
 - storing bounding box of cell.....60
 - Design rules.....*See also* Rules
 - overview.....39
 - overview of theory32
 - DETAIL ON/OFF rule**210**
 - example in quick tutorial.....14
 - overview.....51
 - Detailed logging50, 158, 210, 249, 269, 272
 - finding cause of unpaired error wires.....100
 - Device layers
 - removing from conductive layer.....113
 - Device recognition.....8, 68
 - by containing cell59
 - diagnosing problems.284
 - Device terminals401
 - electical connections112
 - Device wells
 - verifying poor conductors..... 116
 - Diagnosing Mysterious Errors 157
 - Diagnosing problems with rule sets 56, 168
 - Diffusion layer 112
 - example of generation 71
 - Dimension..... *See* Size
 - Dimension verification
 - merged shapes verified 59
 - overview 34
 - Dimensions of shapes
 - using to classify layers..... 64
 - Directional minimum spacing 391
 - Directional spacing checks 89, 254
 - Directory..... 319, 325
 - for output files 361
 - for rules files..... 334
 - long names causing crash 27
 - sample DOS command to change 13, 381
 - Directory creation
 - sample of DOS command to create 13
 - Disk space..... 341
 - effects of ALL_SAFE rule 181
 - scratch directories..... 341
 - Disk swapping
 - caused by ineffective panel size 119
 - minimizing during DRC run..... 119
 - DISPLAY_OPERATIONS cmd line option165, 344
 - Distance
 - classifying shapes by 67
 - classifying shapes by distance apart 235
 - notch and width verification..... 103
 - spacing verification overview..... 84
 - spacing verification theory overview..... 33
 - verifying distance apart 252
 - verifying side length..... 251
 - Distributing a rule set..... 203, 279
 - DO command line option..... 158, 347
 - defining rule sets 295
 - importance of removing from final run 169
 - DOS batch files..... 359
 - DOS commands
 - change directory 13, 360, 381
-

Index

- change drive 13
- copy file..... 13, 381
- create directory..... 13, 381
- editor 15
- find unclosed files 324
- max line length 336
- PAUSE 322, 345
- SET 319, 320, 331, 359
- DRC
 - command line 317
 - defined..... 6
 - diagram of data flow 12
 - overview..... 45
 - overview of steps to execute 318
 - running inside of layout editor 7
 - tips on testing new rules 154
 - troubleshooting..... 27
- DRC command file 365
 - adding commands to..... 356
 - executing in layout editor 368
 - generic overview 52
 - hierarchical output..... 354
 - importing DRC results..... 387
 - selecting output layers 284
 - suppressing macros 357
- DRC command line
 - = disappearing 333
 - adding commands to command file 356
 - adding scales to log file 350
 - ALLOW_QUICK option..... 338
 - border calculation report 348
 - creating in batch file..... 359
 - defined** 329
 - DO rule subsets 347
 - file parameters 334
 - FILESIZE option..... 342
 - HOG option..... 339
 - in advanced tutorial 386
 - in quick tutorial 16
 - input redirection 336
 - list rules file..... 350
 - MAIN_HOG option 340
 - MAIN_MEMORY option 340
 - MAIN_USE option 340
 - max length 336
 - NO_VIRTUAL_MEMORY option..... 340
 - overriding panel border 348
 - overriding wire width 355
 - PAUSE option 322, 345
 - QUICK_PASS option 337
 - QUICK_SPACING option 338
 - reported in log file 362
 - scratch directory 341
 - screen display 343, 344
 - screen display refresh 344
 - SECOND_CELL option..... 335
 - SLOW option 337
 - specifying design area 351
 - specifying input hierarchy 352, 353
 - specifying input hierarchy by num shapes . 353
 - specifying input hierarchy by use count 353
 - specifying layer numbers..... 346
 - specifying output hierarchy 354
 - suppressing rule file warning message 349
 - USE option 339
- DRC layout editor command 16, 159, 318, 334, 382
- DRC log file..... 362
 - short definition 49
- DRC output files 361
- DRC preprocessing..... 58
- DRC rules 69. *See also* Rules
 - adding listing to DRC log..... 350
 - compiled rules file 320, 325
 - file name 319
 - optimization..... 151
- DRC rules compiler
 - command line in tutorial..... 14, 385
 - command line syntax 319
 - error messages 325
 - log file 325
 - output files..... 325
 - sample of log file in quick tutorial..... 15
- DRC_PATH environment variable..... 334, 359
- DRC3CMD.\$\$\$ 362

-
- DRC3-NT.EXE318, 331
 - command line in advanced tutorial386
 - command line in quick tutorial16, 433, 442, 444
 - executing with batch file359
 - DRC3xxx.EXE
 - executable names.....331
 - DRCnAUX.EXE10, 28
 - Drive letter
 - sample DOS command to change.....13
 - Drives
 - using multiple scratch drives342
 - Dummy layers.....36
 - important to verify before final run169
 - removing area from conductive layers113
 - testing154
 - to avoid dangerous operations.....142
 - verification68
 - Edges
 - finding coincident edges.....86
 - Spacing errors mark sides84
 - EDIT layout editor command147, 372, 373
 - EDIT.COM DOS file editor15
 - Electrical connection checks.....8
 - Electrical connections.....110, 402
 - defining200
 - errors due to design area restrictions350
 - finding opens through poor conductors116
 - finding shorts through poor conductors.....308
 - ignored if QUICK_PASS used.....337
 - in MIN_SPACING rules99
 - using in MAX_SPACING rule.....239
 - using in spacing check.....267
 - Emitter layer in bipolar transistors.....114
 - ENCLOSUR.RUL90
 - Enclosure90
 - in MIN_SPACING rule.....85
 - verifying312
 - Enclosure verification
 - overview.....37
 - End caps95, 259
 - END_CMD command line option356
 - example370
 - ENDn keyword of version control rules 176
 - End-to-end sides 97
 - Environment variables
 - long strings cause crash 27
 - Equilateral triangles 226
 - ERR file extension..... 376
 - Error count
 - acute angles not added to..... 77
 - adding shapes on arbitrary layers to 61
 - bad polygons not added to..... 75
 - forcing shapes to be counted as errors..... 398
 - including shapes on arbitrary layer..... 285
 - reported in log file 364
 - terminating when maximum reached 310
 - warning when maximum reached 233
 - ERROR keyword in OUTPUT LAYER rule . 285
 - Error layer..... 61
 - appearance in layout editor..... 366, 370, 371
 - defining 285
 - deleting in layout editor..... 368
 - forcing errors to not be counted 301
 - location of shapes in hierarchical output ... 373
 - overriding width on command line 355
 - selecting in editor 371
 - setting width in rule set..... 315
 - shape coord in command file..... 366
 - shape coord in subcell command file..... 373
 - shape coord in subcell error command file 376
 - shape coordinates in command file..... 366
 - using to force errors..... 213, 228, 240, 308
 - Error marks 386
 - deleting..... 389
 - example in advanced tutorial..... 387
 - example in quick tutorial 18
 - generic overview 52
 - 387
 - setting width in rule set..... 315
 - Error messages
 - preventing console window closing..... 345
 - stored in log file..... 362
 - Error wires
-

Index

- appearance in layout editor366, 370, 371
- cause of unpaired.....100
- finding other wire in pair.....158
- location in hierarchical output.....373
- overriding width on command line.....355
- setting width for all315
- Errors
 - adding boolean results to error count85
 - compiler syntax warnings.....385
 - connection groups413
 - coordinates in command file366
 - coordinates in subcell command file373
 - coordinates in subcell error command file376
 - DANGER logical error message140
 - danger warnings431
 - determining which rule generated a shape.....372
 - executing DRC command file365
 - fixing in layout editor.....372, 388
 - forcing shapes to be counted as errors.....398
 - layout editor tips when fixing.....20
 - merged shapes are checked59
 - messages from rules compiler326
 - Panel is too small to subdivide further127
 - table of rules that do/don't generate errors...62
 - tips for eliminating false errors.....156
 - tips on diagnosing problems.....157
 - use SHOW to find rule that generated shape19
 - using detailed logging to pinpoint50
- ERRORS keyword of OUTPUT LAYER rule453
- ERRWIRE.CMD368
- Escape key.....324, 332
- EXAMPLE1.RUL14
- Examples
 - list of rules files in installation26
- Exclamation mark.....173
- Exclusive OR of layers316
- Executable names331
- Executing command file in the layout editor ..368
- Executing the program with a batch file359
- EXIT layout editor command373
- EXIT layout editor command25
- Expanding layers190
- Export
 - brief overview70
 - example of mask layer418
 - executing command file in layout editor ... 368
 - generic overview of command file52
 - hierarchical output.....146, 354
 - isolating DRC shapes from original data... 369
 - overview of hierarchical output.....134
- Exporting layers
 - overview38
- Extension of one layer past another398
- Fabrication process32
 - accounting for device shrinkage302
 - simulating in rules201
- False errors
 - avoiding false errors for letters.....157
 - avoiding for contact layers87
 - avoiding in MIN_AREA rule108
 - avoiding in MIN_SPACING rule89, 392
 - avoiding with layer manipulation63
 - caused by incorrect dummy layer154
 - caused by limited area checked160
 - diagnosing false shorts112
 - discarding short errors99
 - due to disappearing small shapes.....81
 - due to resolution grid.....80
 - elimination of35
 - MIN_AREA violations caused by other cells138
 - tips for eliminating156
- FET devices
 - electrical connections112
- File editor.....15
- File names
 - long names causing crash27
- Files
 - DRC input files.....334
 - DRC output file names334
 - DRC output files.....361
 - extensions of output files.....361
 - extra commands in command file356
 - file names recorded in log file362
 - hierarchical output.....354

inputs for DRC	330	using to classify layers.....	64
nesting rules files.....	216	Geometric basis	66
overview of data flow.....	12	Geometric basis of a layer	61
rules compiler output files.....	325	Geometric basis of rules	62
FILESIZE command line option.....	162, 342	GEOMETRY keyword of OUTPUT rule	454
should be removed for very large designs ..	167	Getting Started	9
Fill		Grids	79
verifying layer coverage	245	finding off-grid vertices.....	282
Final checklist.....	168	resolution of cut lines	205
Fixing errors	388	snapping vertices to arbitrary grid	304, 306
layout editor tips.....	20	vertices shifted on output.....	132
Flat output		Ground nets	
reported in log file.....	364	finding opens through poor conductors	116
FLATTEN command line option.....	145, 352	Groups of layers.....	111
Flattening	<i>See</i> Hierarchy	HI color.....	371
definition	40	Hierarchical cell command file	147, 374
Flattening cells		HIERARCHICAL command line option	146, 354
ALL_DANGER rule	180	avoiding warning message.....	277
ALL_SAFE rule	181	example in tutorial.....	432
DANGER_CELL rule	207	importing results.....	372
DANGER_LAYER rule.....	209	incompatible with QUICK_PASS	337
SAFE_CELL rule	297	Hierarchical output	
SAFE_LAYER rule.....	299	ALL_DANGER rule	180
Flattening of Cells on Input	144	ALL_SAFE rule	181
Flow of data.....	12	command line option	354
Fundamentals		DANGER_CELL rule	207
Design Rule Verification.....	31	DANGER_LAYER rule	209
Gallium Arsenide technology		deleting results.....	147, 375
air bridges.....	196	diagnosing problems.....	149
Gate layer	112	example in advanced tutorial.....	432
Gate layer testing	401	importing results.....	372
Gate overlaps.....	397	indicated in log file.....	364
Gates		NO_HIER_WARNING rule	277
finding incomplete gates	401	overview	146
Generated CONNECT rules	202	replacing	438
Generated layers	38, 327	SAFE_CELL rule	297
effects of panel processing	131	SAFE_LAYER rule.....	299
hierarchical output.....	134	Hierarchical processing	134, 180, 181, 209
Generating mask layers.....	71	avoiding danger errors.....	141
Generating of mask layers	418	effect of ALL_SAFE	141
Generating Output Layers.....	70	generic overview	44
Geomerty		overview	134

Index

- Hierarchical verification41
- Hierarchy
 - adding hierarchical output as subcells374
 - adding hierarchical output to original cells 373
 - definition40
 - deleting previous hierarchical output375
 - effect on panel processing118
 - flattening cells on input144
 - input hierarchy listed in log file.....363
 - preserved according to number of shapes ..353
 - preserved according to use count353
 - preserved in output.....354
 - preserving in input.....352
 - preserving in input entirely.....353
 - preventing in input entirely352
- HOG command line option.....162, 339
 - for rules compiler321
- HOLE_AREA_FRACTION rule.....**211**
- Holes.....66, 68
 - classifying polygons211
 - finding with ISLANDS rule230
 - in MIN_NOTCH rule.....106
 - locating improperly drawn74
 - removed with BLOAT rule190
 - representation in DRC.....78, 188, 281
- ICED desktop icon11, 384
- ICED™ layout editor..... 8. *See also* layout editor
 - changing appearance of DRC layers ..366, 370
 - executing DRC command file365
 - fixing DRC errors.....372
 - importing DRC layers into284, 368
 - importing hierarchical output372
 - save DRC data in other cell automatically .357
 - selecting error layer shapes371
 - terminating without saving375
 - turning off display of design cell.....371
 - using to execute DRC command16, 382
- ICWIN.BAT382
- Importing DRC layers.....274
- Importing DRC results.....387
- Importing DRC shapes with layout editor368
- IN_CELL rule.....**215**
- INCELL option of INPUT LAYER rule 219, 403
- INCELL processing 59
 - to avoid dangerous operations 142
- INCELL rule
 - effect on hierarchy indicated in log file 363
 - using to identify devices 113
- INCLUDE rule 153, **216**
- Inductors 59
- Input files 47
 - flattening cells 144
 - nesting rules files..... 153
 - overview of data flow..... 12
 - preparation of layout data file..... 16, 382
 - specifying on DRC command line..... 334
- INPUT LAYER rule **217**
 - example in advanced tutorial..... 383
 - example in quick tutorial..... 14
- Input layers 56
- Input Redirection 336
- Installation 10, 11
- Installation directory 11
- Intermediate layers
 - exporting to diagnose problems..... 56
- Interrupting program
 - DRC 332
 - rules compiler..... 324
- Intersecting sides
 - handling specially in spacing check..... 265
- Intersection of layers..... 183, 288, 311, 316
- Introduction 5
- Inverse layer 398
- Inversion of layer 187, 281
- IS_BOX rule **222**
 - reach calculations 126
- IS_CIRCLE rule **225**
- ISLANDS rule 68, **230**
 - ignored if QUICK_PASS used..... 131, 337
- JOURNAL layout editor command 375
- Key
 - used for copy protection 10
- Keywords..... 174, 333
- Large designs

-
- tips for efficient checking.....166
 - Layer 0.....56, 221, 287, 301, 314
 - used to store bounding box of cell.....60
 - using to identify devices.....113
 - using to suppress output.....75
 - Layer Definition
 - detailed overview.....55
 - Layer generation
 - overview.....38
 - Layer Generation Rules.....63
 - LAYER layout editor command.....18, 367, 370
 - Layer manipulation
 - overview.....37
 - Layer names.....172
 - of DRC output layers.....284
 - sample of difference in editor vs. rules.....20
 - syntax restrictions.....55
 - Layer number 99
 - acute angles.....313
 - bad polygons.....189
 - Layer number is also used as input message.....326
 - Layer numbers.....55, 418
 - defining at run time.....57
 - defining input layers.....217
 - defining input/output layers.....273
 - defining output layers.....284
 - listing.....327
 - specifying on command line.....346
 - use care to create unique numbers.....70
 - Layer Processing
 - detailed overview.....55
 - Layers
 - adding output layers to cell file.....366
 - appearance in editor.....366, 371
 - appearance of error wires.....366, 370
 - bloating.....190
 - changing number of in layout editor.....70
 - classifying air bridges.....196
 - classifying by aspect ratio.....184
 - classifying by cell name.....215, 219
 - classifying by circular shape.....225
 - classifying by holes.....211
 - classifying by size.....194, 222
 - classifying by touching other layers .. 288, 311
 - classifying shapes by cell.....59
 - classifying shapes by distance apart.....235
 - classifying with subcell bounding boxes ... 221
 - combing layer numbers on input.....218
 - combining DRC layers into one output layer.....285
 - comparing two designs.....335
 - conductive.....110
 - copying.....187
 - creating inverse.....187, 281
 - defining in advanced tutorial.....383
 - defining input layers.....217
 - defining input/output layers.....273
 - defining output layers.....284
 - deleting layer in layout editor.....368
 - DRC input layers.....217
 - DRC output layers.....284
 - DRC scratch layers.....300
 - etching.....183
 - exclusive OR.....316
 - finding acute angle notches.....231
 - finding acute angles.....242
 - finding all acute angles.....313
 - finding holes.....230
 - finding notches.....248
 - finding shapes less than minimum area.....243
 - finding violations of minimum width.....271
 - flattening hierarchy.....299
 - generating hierarchical output.....354
 - how hierarchical data is generated.....134
 - intersecting.....183
 - inverting.....187
 - list of unconnected.....327
 - listing layers used in rules in DRC log.....363
 - manipulation with DRC rules.....63
 - merging during DRC preprocessing.....58
 - overlap.....288
 - overview of DRC internal layers.....61
 - overview of DRC rules.....63
 - poor conductors.....110, 116
 - removing material from.....113

Index

- replacing in cell.....273
- report on DRC layers in compiler log326
- resolution of cut lines on output205
- retaining hierarchy.....209
- setting width for error wires315
- shrinking.....302
- specifying on command line.....346
- summary of output shapes in log file.....365
- union283
- unused326
- verifying area coverage245
- verifying minimum distance apart252
- verifying minimum side length.....251
- vias and contacts201
- LAYERS command line option.....346
- Layout
 - comparing two designs.....335
 - limiting area checked350
- Layout coordinates of errors362
- Layout data file.....16, 47, 334
 - comparing two designs.....335
 - creation of318
- Layout editor 8. *See also* ICED™ layout editor
 - importing DRC shapes18, 387
 - running DRC inside of7
 - tips when fixing errors.....20
- Layout export to DRC16, 382
- Layout verification
 - overview.....39
- LEFT command line option.....159, 351
- Length
 - using to restrict spacing errors.....268
 - verifying minimum side length.....251
- Length of error shapes
 - discarding short errors.....99
- Letters on mask layers
 - avoiding false errors.....157
 - common cause of errors75
- Level
 - definition40
- Limiting area checked.....159
- LIST_RULES command line option163, 350
- Log file
 - adding border calculations.....348
 - adding rules listing350
 - detailed logging210, 249
 - DRC362
 - listing tolerances in file350
 - rules compiler.....325
 - sample of compiler log file.....15
 - short definition49
- Logical error message.....140
- Long reach rules
 - example440
- LONGCASE command line option165, 343
- LVS utility8
- Macros
 - suppressing in command file357
- Main cell
 - definition40
- Main memory.....162
- MAIN_HOG command line option162, 340
- MAIN_MEMORY command line option.....162, 340
- MAIN_USE command line option162, 340
- Manipulation of layers63
- Manual Organization7
- Mask generation issues74
- MASK keyword of OUTPUT LAYER rule...453
- Mask layer dimensions68
- Mask layers.....63, 70
 - creation of.....284
 - example of generation71
 - generation check list.....82
 - modification of NWELL418
 - output layers tested for acute angles.....313
 - problems with.....74
 - resolution of cut lines on output205
- Mask layers.....*See also* Output Layers; Layers
- Masking
 - dividing input layers.....403
- Masking layers
 - using to identify devices113
- MAX_ANGLE rule231
- MAX_COUNT rule.....154, 233

-
- change warning to automatic termination...310
 - MAX_QUAD rule450
 - MAX_SPACING rule.....**235**
 - brief overview67
 - ignored if QUICK_PASS used.....131, 337
 - Memory
 - conserving with FILESIZE option342
 - effect of border.....127
 - importance of panel size.....119
 - insufficient casuses crash28
 - limiting for DRC339
 - limiting in rules compiler command line....321
 - main vs. data.....340
 - memory available listed in log file363
 - minimizing use of in DRC.....118
 - old virtual memory method340
 - optimizing panel size.....445
 - Memory management161
 - Memory problems
 - solving with smaller panels118
 - try PANEL_VERTICES rule123, 291
 - Memory requirements.....10
 - MERGE layout editor command77
 - Merging of geometry during preprocessing.....58
 - MIN_ANGLE rule.....**242**
 - MIN_AREA rule**243**
 - overview.....107
 - reach.....125
 - using to classify shapes66
 - MIN_FILL rule.....109, **245**
 - MIN_NOTCH rule**248**
 - DRC definition of notch.....106
 - effect of QUICK_PASS130
 - finding other error mark in pair158
 - MIN_SIDE rule108, **251**
 - MIN_SPACING rule**252**
 - angled side exceptions.....93, 256
 - avoiding false errors tutorial391
 - choosing quicker algorithm.....338
 - CONN ignored if QUICK_PASS used.....337
 - defining electrical connections for110, 200
 - detailed logging.....50
 - differences from MAX_SPACING rule 237
 - directional checks 89
 - effect of QUICK_PASS 130
 - example in advanced tutorial..... 383
 - example in quick tutorial..... 14
 - finding other error mark in pair 158
 - important to pair with MIN_NOTCH.. 87, 249
 - overview 84
 - result in quick tutorial..... 20, 23
 - simple examples 88
 - splitting over multiple lines 173
 - MIN_WIDTH rule..... **271**
 - DRC definition of width..... 104
 - example in quick tutorial..... 14
 - finding other error mark in pair 158
 - result in quick tutorial..... 22
 - Minimum dimension verification
 - overview 34
 - Minimum spacing rules
 - overview 84
 - theory overview 33
 - Minimum width rule
 - overview 34
 - Missing rules file
 - suppressing warning 279
 - MODIFY LAYER rule **273**
 - Modify layers..... 57
 - MOSFET technology..... 112
 - MOVE layout editor command..... 21
 - MULTI keyword in STAMP rule 308
 - Multitasking operating systems..... 339
 - keeping console window open..... 322, 345
 - Nested
 - definition 40
 - Nested cells
 - automatic flattening on input 145
 - Nesting.....*See also* Hierarchy
 - Nesting rules files 216
 - Net or node recognition 200
 - Nets
 - definition 110
 - finding opens through poor conductors 116
-

Index

- Networks
 - restrictions for scratch file323, 332
 - users should not share scratch file341
- NFLATTEN command line option145
- NLATTEN command line option353
- NLE utility8
- NO_CHECK_INPUT rule**276**
- NO_FLASH_PANELS cmd line option .165, 344
- NO_FLATTEN command line option145, 353
 - importance in hierarchical output148
- NO_HIER_WARNING rule**277**
- NO_PANELS rule**278**
- NO_RUL command line option349
- NO_RUL rule**279**
- NO_VIRTUAL_MEMORY cmd line option .340
- NO_WARN_ACUTE rule76, **280**
 - remove in final run169
- Node numbers129
 - overview110
- Nodes
 - defined200
- Non-design layers 36. *See also* Dummy layers
 - important to verify before final run169
 - testing154
 - to avoid dangerous operations142
- NONE keyword in STAMP rule308
- NOT keyword in AND rule183
- NOT keyword in IN_CELL rule219
- NOT keywords64
 - simple example64
- NOT rule**281**
- Notation174
- Notches248
 - DRC definition of notch106
 - finding acute angle notches107
 - finding acute angles on specific layer231
 - importance in spacing verification87
 - removed with BLOAT rule190
- NPN transistors
 - sample layer processing114
- Numbers
 - using constants in rules153
- Nwell layer 411
- Nwell layer generation 419
- Obsolete DRC Rules 449
- Octagons 226
- OFF keyword
 - in rule set definition 295
- OFF_GRID rule 81, **282**
- Offsetting a layer 190, 302
- ON keyword
 - in rule set definition 295
- Oops condition 140
- Opens
 - finding opens through poor conductors 116
- Operation number 163
- Optimization of DRC rules 151
- Optimizations
 - tips on increasing speed 161
- Optimizing DRC runs 151
- Optimizing memory usage 161
- Optimizing run time 123, 291
 - separating long reach rules 127
- Optimizing the DRC
 - for large amounts of data 166
- Optional keywords 174
- Options file 336
- OR rule **283**
- Order in rule set 404
- Orientation options in MIN_SPACING rule ... 97
- Outline area 211
- OUTLINE keyword of OUTPUT_LAYER rule 453
- Output
 - isolating DRC shapes from original data... 369
 - save DRC data in other cell automatically. 356
- OUTPUT_ERROR_LAYER 398
 - 387
- Output files 49
 - DRC 361
 - Hierarchical output 146
 - importing DRC output 18, 387
- Output layer
 - using to create mask layer 418

OUTPUT LAYER rule.....	284	Panel is too small to subdivide further.....	127
example in advanced tutorial.....	383	Panel processing	
example in quick tutorial.....	14	border calculations	348
obsolete keywords	453	default behavior.....	121
Output layers	56, 70	generic overview	42
adding DRC layers to cell file	366	overriding border on command line	348
effects of panel processing	131	overview	118
resolution of cut lines	205	Panel size	293
suppressing acute angle check.....	280, 314	listed in log file.....	363
<i>output_file_base_name</i>	49	optimization.....	445
Overlap of one layer past another.....	398	overriding in DRC command line.....	358
OVERLAPPING rule	68, 288	setting by number of vertices per panel.....	122, 290
effect of dangerous processing	139	setting explicitly	123
ignored if QUICK_PASS used.....	337	specifying by maximum area	358
ignored if QUICK_PASS used.....	131	specifying ratio	358
Overlapping shapes		PANEL_VERTICES rule	290
and MIN_SPACING rule.....	84	overview	122
directional spacing checks.....	89	Panels	
MIN_SPACING rule does not always find	252	generated CONNECT rules	202
Overlapping sides	97	PANELX and PANELY rules	293
handling specially in spacing check	263	overriding on DRC command line.....	358
Overlaps		overview	123
electrical connections	111	removing from old rule set	121
Overview of data flow	12	Parallel sides	98
Overview of manual.....	7	exceptions in MIN_SPACING rule	94, 256
Overview of steps to execute DRC	318	Parameters	174
Pads		Pass	
impact of pad rules on speed	441	defined	41, 128
testing minimum spacing of wires from	402	Passes.....	151
writing efficient rules to verify	164	number listed in log file.....	363
Page table	342	overview	128
Panel border.....	124, 440	reasons for varying border.....	127
effect of ASPECT_RATIO rule	185	PATH DOS environment variable	331
effect of HOLE_AREA_FRACTION rule	213	319	
effect of MIN_AREA rule.....	243	PAUSE command line option	322, 345
example	442	345	
overriding	193	PAUSE option	385
recorded in log file	364	PEDIT layout editor command.....	372, 388
reducing execution time	163	Pentagons.....	226
Panel boundaries		Perpendicular edges	
affect output shapes.....	78, 131	avoid marking as MINSPACING errors.....	392
cause of acute angles	423	Perpendicular sides	97

Index

- handling specially in spacing check262
- Physical memory339
- POK file
 - creation of16, 318, 382
- POK file extension.....334
- Polygon layer.....61
- Polygons
 - classifying by aspect ratio184
 - classifying by circular shape225
 - classifying by hole coverage211
 - classifying by size194, 222
 - classifying by touching shapes288, 311
 - classifying shapes by distance apart235
 - finding notches248
 - finding off-grid vertices.....282
 - finding shapes less than minimum area243
 - finding violations of minimum width271
 - merging during DRC preprocessing58
 - removing small polygons303
 - verifying enclosure312
 - verifying minimum distance apart252
 - verifying minimum side length.....251
- Poor conductor layers110, 308, 411
 - finding opens116
- Post-processing of Output Layers78
- Preparing the Binary Layout Data File16, 382
- Preparing the Rules File
 - in advanced tutorial383
 - in quick tutorial14
- Program names331
- Program Requirements10
- Progress reports343
 - overview of reducing run time with.....165
- Protrusions
 - finding acute angle protrusions107
 - in MIN_WIDTH rule104
- P-Select layer
 - example of generation71
- Q:\ICED
 - defined.....11
- QEMM339, 340
- Quick Tutorial12
- QUICK_PASS command line option 164, 337
 - avoiding warning prompt..... 338
 - choice listed in log file 363
 - eliminates electrical tests..... 99
 - important to remove from final run 169
 - overview 129
- QUICK_PASS option
 - avoiding warning prompt..... 182
 - effect on electrical spacing checks 268
- QUICK_SPACING command line option100, 164, 338
 - choice listed in log file 364
 - important to remove from final run 169
- Reach 348
 - definition 124
 - rewriting rules to reduce..... 163
 - specifying in rule 185, 213, 243
- Rectangles..... 64, 222
- RECTANGLES rule 176, 451
- Reducing run times 161
- Redundant DRC rules 151
- Refresh interval of screen display 344
- Removing material from a layer 136
- Reports
 - DRC output files..... 361
 - rules compiler output files 325
- Resistors
 - removing from conductive layer..... 113
 - verifying dummy layer..... 154
- Resolution grids 79
 - avoiding false errors due to 158
 - defining for cut lines..... 205
 - finding off-grid vertices..... 282
 - snapping vertices to arbitrary grid 304, 306
 - vertices shifted on output..... 132
- result_layer*..... 63
- Results 49
- Reusing rule sets 203
- RIGHT command line option 159, 351
- RLO file extension..... 325
- Rule numbers 158
 - determining which rule generated a shape. 366
 - example of reported in compiler log..... 15

-
- executing single rules158
 - listed in command file366
 - listed in compiler log.....327
 - use SHOW to find rule that generated shape19
 - using to execute single rules.....347
 - Rule set order.....404
 - Rule sets**295**
 - evolution during testing.....168
 - executing subsets.....347
 - optimizing for speed.....163
 - tips for organizing153
 - writing for portability203
 - Rule subsets152, 442
 - Rules
 - recommended order.....404
 - adding listing to DRC log.....350
 - automatic optimizations during compilation151
 - compiled file name320, 325
 - compiling319
 - determining which rule generated a shape..372
 - diagnosing problems56
 - DRC command line options346
 - executing single rules158
 - executing subsets.....347
 - execution time per rule.....365
 - file name.....319
 - generic overview39
 - listing in compiler log327
 - listing in DRC log file363
 - nesting files216
 - obsolete syntax449
 - order201, 327
 - reach calculations125
 - splitting over multiple lines172
 - syntax errors.....325
 - syntax overview.....172
 - table of contents69
 - table of geometric and error basis.....62
 - time to execute reported in log file.....163
 - tips on testing new sets.....154
 - Rules compiler.....172, 319
 - automatic optimizations151
 - command line syntax319
 - delays on execution321
 - diagram of data flow.....*See*
 - syntax error warnings385
 - use in advanced tutorial.....383
 - use in quick tutorial14
 - Rules compiler.....*See also* DRC rules compiler
 - Rules file.....47
 - adding to DRC log.....350
 - example in advanced tutorial.....383
 - example in quick tutorial14
 - list of examples in installation26
 - missing source file DRC warning349
 - nesting216
 - recommended oder404
 - search path.....334
 - suppressing warning when missing.....279
 - Rules File Options346
 - Run time
 - bloats can cause excessive.....192
 - decreasing with QUICK_PASS.....130
 - effect of border127
 - effect of panel size.....119
 - importance of panel size.....445
 - important to remove shortcuts from final run169
 - listed in log file.....365
 - optimizing rule sets163
 - tips on reducing161
 - Running the DRC.....317, 329
 - Safe processing.....137
 - effect on hierarchical output.....149
 - options141
 - preventing hierarchical warning prompt....277
 - SAFE_CELL rule**297**
 - overriding for specific layer209
 - overview141
 - SAFE_LAYER rule**299**
 - overview142
 - Sample rules files
 - list of examples in installation26
 - Scale factors
 - listed in log file.....363
-

Index

- Scratch file.....324, 362
 - for rules compiler323
 - for simultaneous runs332
 - left on disk.....332
 - maximum size.....342
 - report in log file.....365
 - size10
 - specifying locations.....341
 - tips for large167
- Scratch layer never used message.....326
- SCRATCH_LAYER rule**300**
- Scratch layers.....57, 287, 300
 - defining in advanced tutorial384
 - exporting to diagnose problems56
- SCRATCH_DIR command line option ..167, 341
- SCRATCH_DIR option
 - for rules compiler command line.....323
- Screen display.....343, 344
- Screen Display Options343
- SECOND_CELL command line option.....335
- SELECT layout editor command.....371
- Selecting error marks in layout editor.....18
- Self-intersecting sides
 - locating.....74
- Separation
 - classifying shapes by distance apart235
 - verifying minimum distance apart252
- SET DOS command359
- Shapes
 - adding output shapes to cell file366
 - appearance of error wires366, 370, 371
 - determining which rule generated a shape..372
 - error coord in subcell error command file..376
 - error coordinates in command file.....366
 - error coordinates in subcell command file .373
 - isolating DRC shapes from original data....369
 - notch and width verification.....103
 - using to classify layers.....64
- Sharp angles.....*See also* Acute angles
 - in mask layers.....76
- Sharp points
 - bloats126
- Shorting layers
 - diagnosing112
- SHORTRUN command line option343
- SHOW layout editor command.....158, 372
- SHOW layout editor command.....19
- SHOW_BORDER command line option.....348
- SHOW_SCALES command line option350
- SHRINK rule68, **302**
 - example274
 - Hierarchical example.....135
 - reach calculations126
 - using to classify wires.....65
- Side length
 - verifying minimum251
- Sides
 - finding coincident edges.....86
 - Spacing errors mark edges.....84
- Simple spacing checks254
- Simultaneous execution332, 362
- Size
 - finding shapes less than minimum area243
 - optimizing DRC for large amounts of data 166
 - removing small polygons.....303
 - using to classify polygons.....184, 194, 222
 - verifying side length.....251
- Size of shapes
 - using to classify layers.....64
- Skewed sides.....79
 - in output shapes.....132
- SKIPPED_POLY rule452
- Slanted sides
 - in output shapes.....132
- Slivers81
- SLOW command line option337
- Slow method
 - algorithm choice listed in log file363
- Small shapes
 - finding by area.....107
- Smooth_tolerance350
- SNAP rule.....158, **304**
- SNAP45 rule.....158, **306**
- Source/Drain layer112, 408

-
- Spacing
 - classifying shapes by distance apart235
 - finding violations of minimum width271
 - verifying minimum distance apart252
 - Spacing method
 - choice listed in log file364
 - Spacing rules
 - overview84
 - theory overview33
 - Spacing verification252
 - overview84
 - verifying serpentine shapes87
 - Spanning rules over multiple lines172
 - Speed
 - fixing slow DRC for small designs339
 - importance of panel size119
 - important to remove shortcuts from final run 169
 - improving by limiting design area350
 - increasing with QUICK_PASS130
 - methods of improving440
 - optimizing for rules compiler321
 - optimizing rule sets163
 - QUICK_PASS option337
 - QUICK_SPACING algorithm338
 - statistics of run listed in log file365
 - tips on increasing161
 - Splitting rules over multiple lines172
 - Squares226
 - STAMP rule116, **308**
 - example in advanced tutorial413
 - ignored if QUICK_PASS used131, 337
 - overview110
 - Stamping node numbers200, 308
 - START_CMD command line option356
 - example370
 - STOP_ON_MAX_COUNT rule154, **310**
 - Storage requirements10
 - Storage problems
 - solving with smaller panels118
 - Subcell bounding boxes
 - storing as shapes221
 - Subcell error command files75, 375
 - bad polygon layer number189
 - Subcells
 - classifying layers by cell215, 219
 - definition40
 - flattening hierarchy181, 297
 - flattening hierarchy of specific layers299
 - generating shapes in430
 - how layers are generated135
 - retaining hierarchy180, 207
 - retaining hierarchy of specific layers209
 - terminating run when error used frequently 233
 - Swap files
 - conserving memory if small162
 - tips for large167
 - SWAP layout editor command70
 - Syntax
 - DRC command line329
 - DRC rules172
 - DRC rules compiler command line319
 - Syntax errors385
 - Rules compiler325
 - Tab characters172
 - TAG366
 - Tag number158
 - using to determine rule that generated shape 19
 - Target Audience6
 - Technical support27
 - try smaller panels first118
 - TEDIT layout editor command372
 - Temporary layers
 - exporting to diagnose problems56
 - Terminating program
 - DRC332
 - rules compiler324
 - Termination
 - automatic when max error count reached .. 310
 - Testing New Rules154
 - Time
 - importance of panel size119
 - listed in log file365
 - tips on reducing161
 - Time stamp

Index

- listed in DRC log file363
- T-intersections97
 - handling specially in spacing check262
- TOEND keyword of version control rules.....176
- Tolerances
 - adding report to log file.....350
 - listed in log file.....363
- TOP command line option.....159, 351
- Touching
 - overview of enclosure verification37
- TOUCHING rule68, **311**
 - adding to MIN_SPACING tests93
 - effect of dangerous processing139
 - example in advanced tutorial.....397
 - example of counting results as errors93
 - ignored if QUICK_PASS used.....131, 337
 - using to divide input layer403
 - using to find coincident edges86
- Touching shapes
 - and MIN_SPACING rule84
 - finding shapes that are not connected.....230
- Touching vs. overlapping288
- Transistor gates.....383
- Transistor wells
 - verifying poor conductors116
- Triangles.....226
- TRIVIAL.CEL.....16
- Troubleshooting.....27
- Tutorial12
 - Advanced379
- Unconnected layers.....327
- Ungrouping *See* Hierarchy
- Ungrouping
 - definition40
- Ungrouping cells on input145
- Ungrouping to prevent danger errors.....140
- Union of layers283
- Unpaired error wires.....268
- UNSELECT layout editor command371
- USE command line option.....162, 339
 - for rules compiler321
- USE layout editor command.....367, 370
- Variable layer numbers..... 57, 346
- Variables..... *See* Constants
 - using constants in rules..... 153
- Version
 - of rules compiler..... 325
 - old versions of the DRC 449
- Version control 176, 178, 179
- Version number of program..... 362
- Versions 331
 - comparing two designs 335
- Vertex shifting
 - in output shapes..... 132
- Vertices
 - shifted due to grid problems..... 79
 - snapping to grid..... 304, 306
- Vias..... 87, 201
 - in CONNECT rule..... 111
- VIEW layout editor command 371
- Violations *See* Errors
- Virtual array page table 162
- Virtual memory..... 339, 341
 - efficiency listed in log file..... 365
 - max size..... 342
 - old method..... 340
 - optimizing DRC for large amounts of data 166
- WARN_ACUTE rule..... **313**
- WARN_ACUTE=0 rule
 - important to remove from final run 169
- Warning messages
 - default panel size..... 121
 - stored in log file..... 362
 - window closing too soon 322, 345
- Warning prompt
 - avoiding..... 338, 349
- Warnings
 - from rules compiler 326
- Well layer 411
 - verifying poor conductors..... 116
- Well layer generation..... 419
- Well shapes
 - finding shorts through poor conductors..... 308
- Whitespace characters 172

-
- Width
- default for error wires.....315, 355
 - DRC definition103
 - finding violations of minimum271
 - of error wires366, 370
387
- Width of wires
- classifying by width.....65
- Window
- closing console window322, 345
- Wire layer61
- Wire type366, 370
- WIRE_WIDTH command line option.....355
- WIRE_WIDTH rule **315**
- overriding on command line..... 355
- Wires
- appearance in layout editor..... 366, 370, 371
 - classifying by width..... 65
 - converted to polygons during preprocessing 58
 - creation in output data 286
 - finding spacing errors in serpentine shapes . 87
 - overriding width on command line..... 355
 - setting width for error wires 315
- XOR rule **316**