Magic Tutorial #6: Design-Rule Checking

John Ousterhout

Computer Science Division
Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

(Updated by others, too.)

This tutorial corresponds to Magic version 7.

Tutorials to read first:

Magic Tutorial #1: Getting Started
Magic Tutorial #2: Basic Painting and Selection
Magic Tutorial #4: Cell Hierarchies

Commands introduced in this tutorial:

:drc

Macros introduced in this tutorial:

y

1 Continuous Design-Rule Checking

When you are editing a layout with Magic, the system automatically checks design rules on your behalf. Every time you paint or erase, and every time you move a cell or change an array structure, Magic rechecks the area you changed to be sure you haven't violated any of the layout rules. If you do violate rules, Magic will display little white dots in the vicinity of the violation. This error paint will stay around until you fix the problem; when the violation is corrected, the error paint will go away automatically. Error paint is written to disk with your cells and will re-appear the next time the cell is read in. There is no way to get rid of it except to fix the violation.

Continuous design-rule checking means that you always have an up-to-date picture of design-rule errors in your layout. There is never any need to run a massive check over the whole design unless you change your design rules. When you make small changes to an existing layout, you will find out immediately if you've introduced errors, without having to completely recheck the entire layout.

To see how the checker works, run Magic on the cell **tut6a**. This cell contains several areas of metal (blue), some of which are too close to each other or too narrow. Try painting and erasing metal to make the error paint go away and re-appear again.

2 Getting Information about Errors

In many cases, the reason for a design-rule violation will be obvious to you as soon as you see the error paint. However, Magic provides several commands for you to use to find violations and figure what's wrong in case it isn't obvious. All of the design-rule checking commands have the form

:drc *option*

where *option* selects one of several commands understood by the design-rule checker. If you're not sure why error paint has suddenly appeared, place the box around the error paint and invoke the command

:drc why

This command will recheck the area underneath the box, and print out the reasons for any violations that were found. You can also use the macro **y** to do the same thing. Try this on some of the errors in **tut6a**. It's a good idea to place the box right around the area of the error paint: **:drc why** rechecks the entire area under the box, so it may take a long time if the box is very large.

If you're working in a large cell, it may be hard to see the error paint. To help locate the errors, select a cell and then use the command

:drc find [*nth*]

If you don't provide the *nth* argument, the command will place the box around one of the errors in the selected cell, and print out the reason for the error, just as if you had typed **:drc why**. If you invoke the command repeatedly, it will step through all of the errors in the selected cell. (remember, the "." macro can be used to repeat the last long command; this will save you from having to retype **:drc find** over and over again). Try this out on the errors in **tut6a**. If you type a number for *nth*, the command will go to the *nth* error in the selected cell, instead of the next one. If you invoke this command with no cell selected, it searches the edit cell.

A third drc command is provided to give you summary information about errors in hierarchical designs. The command is

:drc count

This command will search every cell (visible or not) that lies underneath the box to see if any have errors in them. For each cell with errors, **:drc count** will print out a count of the number of error areas.

3 Errors in Hierarchical Layouts

The design-rule checker works on hierarchical layouts as well as single cells. There are three overall rules that describe the way that Magic checks hierarchical designs:

1. The paint in each cell must obey all the design rules by itself, without considering the paint in any other cells, including its children.

2. The combined paint of each cell and all of its descendants (subcells, sub-subcells, etc.) must be consistent. If a subcell interacts with paint or with other subcells in a way that introduces a design-rule violation, an error will appear in the parent. In designs with many levels of hierarchy, this rule is applied separately to each cell and its descendants.

3. Each array must be consistent by itself, without considering any other subcells or paint in its parent. If the neighboring elements of an array interact to produce a design-rule violation, the violation will appear in the parent.

To see some examples of interaction errors, edit the cell **tut6b**. This cell doesn't make sense electrically, but illustrates the features of the hierarchical checker. On the left are two subcells that are too close together. In addition, the subcells are too close to the red paint in the top-level cell. Move the subcells and/or modify the paint to make the errors go away and reappear. On the right side of **tut6b** is an array whose elements interact to produce a design-rule violation. Edit an element of the array to make the violation go away. When there are interaction errors between the elements of an array, the errors always appear near one of the four corner elements of the array (since the array spacing is uniform, Magic only checks interactions near the corners; if these elements are correct, all the ones in the middle must be correct too).

It's important to remember that each of the three overall rules must be satisfied independently. This may sometimes result in errors where it doesn't seem like there should be any. Edit the cell **tut6c** for some examples of this. On the left side of the cell there is a sliver of paint in the parent that extends paint in a subcell. Although the overall design is correct, the sliver of paint in the parent is not correct by itself, as required by the first overall rule above. On the right side of **tut6c** is an array with spacing violations between the array elements. Even though the paint in the parent masks some of the problems, the array is not consistent by itself so errors are flagged. The three overall rules are more conservative than strictly necessary, but they reduce the amount of rechecking Magic must do. For example, the array rule allows Magic to deduce the correctness of an array by looking only at the corner elements; if paint from the parent had to be considered in checking arrays, it would be necessary to check the entire array since there might be parent paint masking some errors but not all (as, for example, in **tut6c**).

Error paint appears in different cells in the hierarchy, depending on what kind of error was found. Errors resulting from paint in a single cell cause error paint to appear in that cell. Errors resulting from interactions and arrays appear in the parent of the interacting cells or array. Because of the way Magic makes interaction checks, errors can sometimes "bubble up" through the hierarchy and appear in multiple cells. When two cells overlap, Magic checks this area by copying all the paint in that area from both cells (and their descendants) into a buffer and then checking the buffer. Magic is unable to tell the difference between an error from one of the subcells and an error that comes about because the two subcells overlap incorrectly. This means that errors in an interaction area of a cell may also appear in the cell's parent. Fixing the error in the subcell will cause the error in the parent to go away also.

4 Turning the Checker Off

We hope that in most cases the checker will run so quickly and quietly that you hardly know it's there. However, there will probably be some situations where the checker is irksome. This section describes several ways to keep the checker out of your hair.

If you're working on a cell with lots of design-rule violations the constant redisplay caused by design-rule checking may get in your way more than it helps. This is particularly true if you're in the middle of a large series of changes and don't care about design-rule violations until the changes are finished. You can stop the redisplay using the command

:see no errors

After this command is typed, design-rule errors will no longer be displayed on the screen. The design-rule checker will continue to run and will store error information internally, but it won't bother you by displaying it on the screen. When you're ready to see errors again, type

:see errors

There can also be times when it's not the redisplay that's bothersome, but the amount of CPU time the checker takes to recheck what you've changed. For example, if a large subcell is moved to overlap another large subcell, the entire overlap area will have to be rechecked, and this could take several minutes. If the prompt on the text screen is a "]" character, it means that the command has completed but the checker hasn't caught up yet. You can invoke new commands while the checker is running, and the checker will suspend itself long enough to process the new commands.

If the checker takes too long to interrupt itself and respond to your commands, you have several options. First, you can hit the interrupt key (often ˆC) on the keyboard. This will stop the checker immediately and wait for your next command. As soon as you issue a command or push a mouse button, the checker will start up again. To turn the checker off altogether, type the command

:drc off

From this point on, the checker will not run. Magic will record the areas that need rechecking but won't do the rechecks. If you save your file and quit Magic, the information about areas to recheck will be saved on disk. The next time you read in the cell, Magic will recheck those areas, unless you've still got the checker turned off. There is nothing you can do to make Magic forget about areas to recheck; **:drc off** merely postpones the recheck operation to a later time.

Once you've turned the checker off, you have two ways to make sure everything has been rechecked. The first is to type the command

:drc catchup

This command will run the checker and wait until everything has been rechecked and errors are completely up to date. When the command completes, the checker will still be enabled or disabled just as it was before the command. If you get tired of waiting for **:drc catchup**, you can always hit the interrupt key to abort the command; the recheck areas will be remembered for later. To turn the checker back on permanently, invoke the command

:drc on

5 Porting Layouts from Other Systems

You should not need to read this section if you've created your layout from scratch using Magic or have read it from CIF using Magic's CIF or Calma reader. However, if you are bringing into Magic a layout that was created using a different editor or an old version of Magic that didn't have continuous checking, read on. You may also need to read this section if you've changed the design rules in the technology file.

In order to find out about errors in a design that wasn't created with Magic, you must force Magic to recheck everything in the design. Once this global recheck has been done, Magic will use its continuous checker to deal with any changes you make to the design; you should only need to do the global recheck once. To make the global recheck, load your design, place the box around the entire design, and type

:drc check

This will cause Magic to act as if the entire area under the box had just been modified: it will recheck that entire area. Furthermore, it will work its way down through the hierarchy; for every subcell found underneath the box, it will recheck that subcell over the area of the box.

If you get nervous that a design-rule violation might somehow have been missed, you can use **:drc check** to force any area to be rechecked at any time, even for cells that were created with Magic. However, this should never be necessary unless you've changed the design rules. If the number of errors in the layout ever changes because of a **:drc check**, it is a bug in Magic and you should notify us immediately.