Reduced Magic Tutorial

This reduced tutorial consists of excerpts from the tutorials presented in "WRL Research Report 90/7 — 1990 DECWRL/Livermore Magic Release" by R.N. Mayo, M.H. Arnold, W.S. Scott, D. Stark, and G.T. Hamachi (and J. Ousterhout). The required copyright is as follows:

1. What is Magic

Magic is an interactive system for creating and modifying VLSI circuit layouts. With Magic, you use a color graphics display and a mouse to design basic cells and to combine them hierarchically into larger structures. Magic is different from other layout editors you may have used. The most important difference is that Magic is more than just a color painting tool: it understands quite a bit about the nature of circuits and uses this information to provide you with additional operations. For example, Magic has built-in knowledge of layout rules; as you are editing, it continuously checks for rule violations. Magic also knows about connectivity and transistors, and contains a built-in hierarchical circuit extractor. Magic also has a plow operation that you can use to stretch or compact cells. Lastly, Magic has routing tools that you can use to make the global interconnections in your circuits.

In ELE334, we will only be using the color painting tool and the design-rule checker.

2. Getting Help

The *man* page gives concise and complete descriptions of all the Magic commands. However, the *man* page may not make much sense until after you've read this tutorial.

A second way of getting help is available on-line through Magic itself. The **:help** command will print out one line for each Magic command, giving the command's syntax and an extremely brief description of the command. This facility is useful if you've forgotten the name or exact syntax of a command. If you're interested in information about a particular subject, you can type

:help *subject*

This command will print out each command description that contains the *subject* string.

3. Running Magic

From this point on, you should be sitting at a Magic workstation so you can experiment with the program as you read this tutorial. Starting up Magic is usually pretty simple. Just log in and, if needed, start up your X-windows. Then type the shell command

magic tut1

Tut1 is the name of a library cell that you will play with. At this point, several colored rectangles should appear on the color display along with a white box and a cursor. A message will be printed on the text display to tell you that **tut1** isn't writable (it's in a read-only library), and a ">" prompt should appear.

4. The Box and the Cursor

Two things, called the box and the cursor, are used to select things on the color display. As you move the mouse, the cursor moves on the screen. The cursor starts out with a crosshair shape, but its shape changes as you work to provide feedback about what you're doing. The left and right mouse buttons are used to position the box. If you press the left mouse button and then release it, the box will move so that its lower left corner is at the cursor position. If you press and release the right mouse button, the upper right corner of the box will move to the cursor position, but the lower left corner will not change. These two buttons are enough to position the box anywhere on the screen.

Try using the buttons to place the box around each of the colored rectangles on the screen.

5. Invoking Commands

Commands can be invoked in Magic in three ways: by pressing buttons on the mouse; by typing single keystrokes on the text keyboard (these are called *macros*); or by typing longer commands on the text keyboard (these are called *long commands*). Many of the commands use the box and cursor to help guide the command.

To see how commands can be invoked from the buttons, first position the box over a small blank area in the middle of the screen. Then move the cursor over the red rectangle and press the middle mouse button. At this point, the area of the box should get painted red. Now move the cursor over empty space and press the middle button again. The red paint should go away. Note how this command uses both the cursor and box locations to control what happens.

As an example of a macro, type the **g** key on the text keyboard. A grid will appear on the color display, along with a small black box marking the origin of the cell. If you type g again, the grid will go away. You may have noticed earlier that the box corners didn't move to the exact cursor position: you can see now that the box is forced to fall on grid points.

Long commands are invoked by typing a colon (“:”) or semi-colon (“;”). After you type the colon or semi-colon, the “>” prompt on the text screen will be replaced by a “:” prompt. This indicates that Magic is waiting for a long command. At this point you should type a line of text, followed by a return. When the long command has been processed, the “>” prompt reappears on the text display. Try typing semi-colon followed by return to see how this works. Occasionally a “]” (right bracket) prompt will appear. This means that the design-rule checker is reverifying part of your design. For now you can just ignore this and treat “]” like “>”.

Each long command consists of the name of the command followed by arguments, if any are needed by that command. The command name can be abbreviated, just as long as you type enough characters to distinguish it from all other long commands. For example, **:h** and **:he** may be used as abbreviations for **:help**. On the other hand, **:u** may not be used as an abbreviation for **:undo** because there is another command, **:upsidedown**, that has the same abbreviation. Try typing **:u**.

As an example of a long command, put the box over empty space on the color display, then invoke the long command

:paint red

The box should fill with the red color, just as if you had used the middle mouse button to paint it. Everything you can do in Magic can be invoked with a long command. It turns out that the macros are just conveniences that are expanded into long commands and executed. For example, the long

command equivalent to the **g** macro is

:grid

One more long command is of immediate use to you. It is

:quit

Invoke this command. Note that before exiting, Magic will give you one last chance to save the information that you've modified. Type **y** to exit without saving anything.

6. Painting and Erasing

Enter Magic to edit the cell **tut2a** (type **magic tut2a** to the Unix shell). The **tut2a** cell is a sort of palette: it shows a splotch of each of several paint layers and gives the names that Magic uses for the layers.

The two basic layout operations are painting and erasing. They can be invoked using the **:paint** and **:erase** long commands, or using the buttons. The easiest way to paint and erase is with the mouse buttons. To paint, position the box over the area you'd like to paint, then move the cursor over a color and click the middle mouse button. To erase everything in an area, place the box over the area, move the cursor over a blank spot, and click the middle mouse button. Try painting and erasing various colors. If the screen gets totally messed up, you can always exit Magic and restart it. While you're painting, white dots may occasionally appear and disappear. These are design rule violations detected by Magic, and will be explained in the section 14. You can ignore them for now.

It's completely legal to paint one layer on top of another. When this happens, one of three things may occur. In some cases, the layers are independent, so what you'll see is a combination of the two, as if each were a transparent colored foil. This happens, for example, if you paint metal1 (blue) on top of polysilicon (red). In other cases, when you paint one layer on top of another you'll get something different from either of the two original layers. For example, painting poly on top of ndiff produces ntransistor (try this). In still other cases the new layer replaces the old one: this happens, for example, if you paint a pcontact on top of ntransistor. Try painting different layers on top of each other to see what happens. The meaning of the various layers is discussed in more detail in Section 12 below.

There is a second way of erasing paint that allows you to erase some layers without affecting others. This is the macro **^D** (control-D, for ``**D**elete paint''). To use it, position the box over the area to be erased, then move the crosshair over a splotch of paint containing the layer(s) you'd like to erase. Type **^D** key on the text keyboard: the colors underneath the cursor will be erased from the area underneath the box, but no other layers will be affected. Experiment around with the **^D** macro to try different combinations of paints and erases. If the cursor is over empty space then the **^D** macro is equivalent to the middle mouse button: it erases everything.

You can also paint and erase using the long commands

:paint *layers*
:erase *layers*

In each of these commands layers is one or more layer names separated by commas (you can also use spaces for separators, but only if you enclose the entire list in double-quotes). Any layer can be abbreviated as long as the abbreviation is unambiguous. For example, **:paint poly,metal1** will paint the polysilicon and metal1 layers. The macro **^D** is predefined by Magic to be **:erase $** (**$** is a pseudo-layer that means ``all layers underneath the cursor'').

7. Undo

There are probably going to be times when you'll do things that you'll later wish you hadn't. Fortunately, Magic has an undo facility that you can use to restore things after you've made mistakes. The command

:undo

(or, alternatively, the macro **u**) will undo the effects of the last command you invoked. If you made a mistake several commands back, you can type **:undo** several times to undo successive commands. However, there is a limit to all this: Magic only remembers how to undo the last ten or so commands. If you undo something and then decide you wanted it after all, you can undo the undo with the command

:redo

(**U** is a macro for this command). Try making a few paints and erases, then use **:undo** and **:redo** to work backwards and forwards through the changes you made.

8. Selecting Objects

Once you have painted a piece of layout, there are several commands you can invoke to modify the layout. Many of them are based on the *selection*: you select one or more pieces of the design, and then perform operations such as copying, deletion, and rotation on the selected things. To see how the selection works, load cell **tut2b**.

The first thing to do is to learn how to select. Move the cursor over the upper portion of the L-shaped blue area in **tut2b**, and type **s**, which is a macro for **:select**. The box will jump over to cover the vertical part of the “L”. This operation selected a chunk of material. Move the box away from the chunk, and you'll see that a thin white outline is left around the chunk to show that it's selected. Now move the cursor over the vertical red bar on the right of the cell and type **s**. The box will move over that bar, and the selection highlighting will disappear from the blue area.

If you type **s** several times without moving the cursor, each command selects a slightly larger piece of material. Move the cursor back over the top of the blue “L”, and type **s** three times without moving the cursor. The first **s** selects a chunk (a rectangular region all of the same type of material). The second **s** selects a region (all of the blue material in the region underneath the cursor, rectangular or not). The third **s** selects a net (all of the material that is electrically connected to the original chunk; this includes the blue metal, the red polysilicon, and the contact that connects them).

The macro **S** (short for **:select more**) is just like **s** except that it adds on to the selection, rather than replacing it. Move the cursor over the vertical red bar on the right and type **S** to see how this works. You can also type **S** multiple times to add regions and nets to the selection.

If you accidentally type **s** or **S** when the cursor is over space, you'll select a cell (**tut2b** in this case). You can just undo this. The use of hierarchical cells will not be discussed in this tutorial.

You can also select material by area: place the box around the material you'd like to select and type **a** (short for **:select area**). This will select all of the material underneath the box. You can use the macro **A** to add material to the selection by area, and you can use the long command

:select [more] area *layers*

to select only material on certain layers. Place the box around everything in **tut2b** and type **:select area metal1** followed by **:select more area poly**.

If you'd like to clear out the selection without modifying any of the selected material, you can use the command

:select clear

or type the macro **C**. You can clear out just a portion of the selection by typing **:select less** or **:select less area** *layers*; the former deselects paint in the order that **:select** selects paint, while the latter deselects paint under the box (just as **:select area** selects paint under the box). For a synopsis of all the options to the **:select** command, type

:select help

9. Operations on Selected Objects

Once you've made a selection, there are a number of operations you can perform on it:

:delete
:move [*direction* [*distance*]]
:stretch [*direction* [*distance*]]
:copy
:upsidedown
:sideways
:clockwise [*degrees*]

The **:delete** command deletes everything that's selected. Watch out: **:delete** is different from **:erase**, which erases paint from the area underneath the box. Select the red bar on the right in **tut2b** and type **d**, which is a macro for **:delete**. Undo the deletion with the **u** macro.

The **:move** command picks up both the box and the selection and moves them so that the lower-left corner of the box is at the cursor location. Select the red bar on the right and move it so that it falls on top of the vertical part of the blue "L". You can use **t** ("translate") as a macro for **:move**. Practice moving various things around the screen. The command **:copy** and its macro **c** are just like **:move** except that a copy of the selection is left behind at the original position.

There is also a longer form of the **:move** command that you can use to move the selection a precise amount. For example, **:move** up 10 will move the selection (and the box) up 10 units. The direction argument can be any direction like **left**, **south**, **down**, etc. The macros **q**, **w**, **e**, and **r** are defined to move the selection left, down, up, and right (respectively) by one unit.

The **:stretch** command is similar to **:move** except that it stretches and erases as it moves. **:stretch** does not operate diagonally, so if you use the cursor to indicate where to stretch to, Magic will either stretch up, down, left, or right, whichever is closest. The **:stretch** command moves the selection and also does two additional things. First, for each piece of paint that moves, **:stretch** will erase that layer from the region that the paint passes through as it moves, in order to clear material out of its way. Second, if the back edge of a piece of selected paint touches non-selected material, one of the two pieces of paint is stretched to maintain the connection. The macros **Q**, **W**, **E**, and **R** just like the macros **q**, etc. described above for **:move**. The macro **T** is predefined to **:stretch**. To see how stretching works, select the horizontal piece of the green wire in **tut2b** and type **W**, then **E**. Stretching only worries about material in front of and behind the selection; it ignores material to the sides (try the **Q** and **R** macros to see).

The command **:upsidedown** will flip the selection upside down, and **:sideways** flips the selection sideways. Both commands leave the selection so it occupies the same total area as before, but with the contents flipped. The command **:clockwise** will rotate the selection clockwise, leaving

the lower-left corner of the new selection at the same place as the lower-left corner of the old selection. Degrees must be a multiple of 90, and defaults to 90.

At this point you know enough to do quite a bit of damage to the **tut2b** cell. Experiment with the selection commands. Remember that you can use **:undo** to back out of trouble.

10. File Saving

Magic saves cells in a special Magic format where the name of the file is just the name of the cell with **.mag** appended. For example, the cell **tut2a** is saved in file **tut2a.mag**. To save cells on disk, invoke the command

:save *name*

This command will append ".mag" to *name* and save the cell you are editing in that location.

Once a cell has been saved on disk you can edit it by invoking Magic with the command

magic *name*

where *name* is the same name you used to save the cell (no ".mag" extension).

11. Utility Commands

There are several additional commands that you will probably find useful once you start working on real cells. The command

:grid [*spacing*]
:grid off

will display a grid over your layout. Initially, the grid has a one-unit spacing. Typing **:grid** with no arguments will toggle the grid on and off. If a single numerical argument is given, the grid will be turned on, and the grid lines will be *spacing* units apart. The macro **g** provides a short form for **:grid** and **G** is short for **:grid 2**. The command **:grid off** always turns the grid off, regardless of whether or not is was previously on. When the grid is on, a small black box is displayed to mark the (0,0) coordinate of the cell you're editing.

If you want to create a cell that doesn't fit on the screen, you'll need to know how to change the screen view. This can be done with three commands:

:zoom *factor*
:findbox [**zoom**]
:view

If *factor* is given to the zoom command, it is a zoom-out factor. For example, the command **:zoom 2** will change the view so that there are twice as many units across the screen as there used to be (**Z** is a macro for this). The new view will have the same center as the old one. The command **:zoom .5** will increase the magnification so that only half as much of the circuit is visible.

The **:findbox** command is used to change the view according to the box. The command alone just moves the view (without changing the scale factor) so that the box is in the center of the screen. If the zoom argument is given then the magnification is changed too, so that the area of the box nearly fills the screen. **z** is a macro for **:findbox zoom** and **B** is a macro for **:findbox**.

The command **:view** resets the view so that the entire cell is visible in the window. It comes in handy if you get lost in a big layout. The macro **v** is equivalent to **:view**.

IMPORTANT: The grid will not be visible if the view is too great such that the grid lines

would become indistinguishable.

The command **:box** prints out the size and location of the box in case you'd like to measure something in your layout. The macro **b** is predefined to **:box**. The **:box** command can also be used to set the box to a particular location, height, or width.

The command

:what

will print out information about what's selected. This may be helpful if you're not sure what layer a particular piece of material is, or what layer a particular label is attached to.

12. What the Layers Mean in Magic

The paint layers available in Magic are different from those in other systems because they don't correspond exactly to the masks used in fabrication. We call them abstract layers because they correspond to constructs such as wires and contacts, rather than mask layers. We also call them logs because they look like sticks except that the geometry is drawn fully fleshed instead of as lines. In Magic there is one paint layer for each kind of conducting material (polysilicon, ndiffusion, metal1, etc.), plus one additional paint layer for each kind of transistor (ntransistor, ptransistor, etc.), and, finally, one further paint layer for each kind of contact (pcontact, ndcontact, m2contact, etc.) Each layer has one or more names that are used to refer to that layer in commands. To find out the layers available in the current technology, type the command

:layers

An advantage of the logs used in Magic is that they simplify the design rules. Most of the formation rules (e.g. contact structure) go away, since Magic automatically generates correctly-formed structures. All that are left are minimum size and spacing rules, and Magic's abstract layers result in fewer of these than there would be otherwise. This helps to make Magic's built-in design rule checker very fast, and is one of the reasons plowing is possible.

13. Multiple Windows

A window is a rectangular viewport. You can think of it as a magnifying glass that may be moved around on your chip. Magic initially displays a single window on the screen. This section will show you how to create new windows. Multiple windows allow you to view several portions of a circuit at the same time, or even portions of different circuits.

13.1. Manipulating windows

Initially Magic displays one large window. The

:openwindow [*cellname*]

command opens another window and loads the given cell. To give this a try, start up Magic with the command magic **tut5a**. Then point anywhere in a Magic window and type the command **:openwindow tut5b** (make sure you're pointing to a Magic window). A new window will appear and it will contain the cell **tut5b**. If you don't give a *cellname* argument to **:openwindow**, it will open a new window on the cell containing the box, and will zoom in on the box. The macro **o** is predefined to **:openwindow**. Try this out by placing the box around an area of **tut5b** and then typing **o**. Another window will appear. You now have three windows, all of which display pieces of layout. Magic doesn't care how many windows you have (within reason) nor how they overlap.

To get rid of a window, point to it and type

:closewindow

or use the macro **O**. Point to a portion of the original window and close it.

13.2. How commands work inside of windows

Each window has a caption at the top. Here is an example:

mychip EDITING

This indicates that the window contains the root cell **mychip** and is being edited. At any given time Magic is editing exactly one cell. If the edit cell is in another window then the caption on this window will read:

mychip [NOT BEING EDITED]

Let's do an example to see how commands are executed within windows. Close any layout windows that you may have on the screen and open two new windows, each containing the cell **tut5a**. (Use the **:closewindow** and **:openwindow tut5a** commands to do this.) Try moving the box around in one of the windows. Notice that the box also moves in the other window. Windows containing the same root cell are equivalent as far as the box is concerned: if it appears in one it will appear in all, and it can be manipulated from them interchangeably. If you change **tut5a** by painting or erasing portions of it you will see the changes in both windows. This is because both windows are looking at the same thing: the cell **tut5a**. Go ahead and try some painting and erasing until you feel comfortable with it. Try positioning one corner of the box in one window and another corner in another window. You'll find it doesn't matter which window you point to, all Magic knows is that you are pointing to **tut5a**.

We have seen how Magic behaves when both windows view a single cell. What happens when windows view different cells? To try this out load **tut5b** into one of the windows (point to a window and type **:load tut5b**). You will see the captions on the windows change - only one window contains the cell currently being edited. The box cannot be positioned by placing one corner in one window and another corner in the other window because that doesn't really make sense (try it). However, the selection commands work between windows: you can select information in one window and then copy it into another. This only works if the window you're copying into contains the edit cell; if not, you'll have to *close the edit window* and use the command

:load *name*

The **:load** *name* loads the cell *name* into the window underneath the cursor.

The operation of many Magic commands is dependent upon which window you are pointing at. If you are used to using Magic with only one window you may, at first, forget to point to the window that you want the operation performed upon. For instance, if there are several windows on the screen you will have to point to one before executing a command like **:grid** - otherwise you may not affect the window that you intended!

14. 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.

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.

15. The SCMOS Technology

The most important characteristic of the SCMOS technology is that it is flavor-less and scalable: layouts designed using the SCMOS rules may be fabricated in either N-well or P-well technology at a variety of feature sizes The lambda units used in Magic are dimensionless and can be scaled to dimensions such as 0.6 microns/lambda, 1.0 microns/lambda, 1.5 microns/ lambda, etc. In order for SCMOS designs to be fabricated with either N-well or P-well technology, both p-well and n-well contacts must be placed, and where wells and rings are specified explicitly (e.g. in pads) both flavors must be specified. When the circuit is fabricated, one of the flavors of wells, rings, and substrate contacts will be ignored.

The SCMOS technology provides two levels of metal. All contacts are to first-level metal. There are no buried or stacked contacts (mushroom contacts).

Note that contacts are drawn in terms of the total overlap area between the layers being connected, not in terms of the via holes.

15.1. Layers and design rules

Second-level metal

The top level of metal is drawn in a purple color, and has the names metal2 or m2 or purple. It must always be at least 3 units wide, and metal2 areas must be separated from each other by at least 4 units.

First-level metal

The lower level of metal is drawn in blue and has the names metal1 or m1 or blue. It has a minimum width of 3 units and a minimum spacing of 3 units.

Polysilicon

Polysilicon is drawn in red, and can be referred to in Magic as either polysilicon or red or poly or p. It has a minimum width of x_{p1} units and a minimum spacing of x_{p2} units.

Diffusion

In the SCMOS technology, it is unnecessary (however it is recommended) for you to specify wells and implant selection layers explicitly. Instead, there are four different layers that correspond to the two kinds of diffusion in the two kinds of wells. Based on these four layers, Magic automatically generates the masks for active, wells, and implant select.

The most common kinds of diffusion are p-diffusion in an n-well (n-substrate) and n-diffusion in a p-well (p-substrate); they are used for creating p-type and n-type transistors, respectively. P-diffusion in an n-well is drawn with a light brown color; Magic accepts the names pdiffusion, pdiff, and brown for this layer. N-diffusion in a p-well is drawn in green, and can be referred to as ndiffusion, ndiff, or green.

The other two kinds of diffusion are used for generating well (substrate) contacts and guard rings; they consist of a strongly implanted diffusion area in a well of the same type. P-diffusion in a p-well is drawn with a light brown color and has stippled holes in it. It goes by the names psubstratepdiff, psd, ppdiff, ppd, or pohmic. N-diffusion in an n-well is drawn in a light green color with stippled holes in it, and goes by the names nsubstratendiff, nsd, nndiff, nnd or nohmic.

The basic design rules for the first two kinds of diffusion are the same: they must be at least x_{d1} units wide and have a spacing (to the same kind of diffusion) of at least x_{d2} units. The second kinds of diffusion used as substrate (well) contacts must be at least 4 units wide and have a spacing (to the same kind of diffusion) of at least 3 units. Spacing rules between diffusions of different types are discussed below.

Metal 2 contacts

All contacts involve the metal1 layer. In Magic, contacts aren’t drawn as two areas of overlapping material with a via hole in the middle. Instead, you just draw a single large area of a special contact type, m2contact in the case of metal2 contacts. This corresponds to the area where the two wiring layers overlap (metal and metal2 in the case of metal2 contacts). Magic will automatically output metal1, metal2 and will also generate the small via hole in the center of the contact area. All contacts must be rectangular: two contacts of the same type may not abut. Contacts from metal1 to metal2 are called m2contact, m2cut, m2c, via, or v. They appear on the screen as an area of metal1 overlapping an area of metal2, with a black waffle pattern over the contact area. Metal2 contacts must be at least 4 units wide.

There is an additional special rule for metal2 contacts: there must not be any polysilicon or diffusion edges underneath the area of the contact or within 1 unit of the contact. This rule is present because it is hard to fabricate a metal2 contact over the sharp rise of a poly or diffusion edge. It is acceptable for poly or diffusion to lie under a m2contact area, as long as it completely covers the area of the m2contact with an additional 1-unit surround.

Polysilicon and diffusion contacts

Contacts between metal1 and polysilicon go by the names polycontact, pcontact, polycut, and pc. As with all contacts, you only draw the outer boundary of the overlapping area of poly and metal1; Magic fills in the contact cut(s) at mask generation time. Pcontact areas must be at least 4 units wide and must be separated from unrelated polysilicon and pcontact by at least 3 units. Pcontacts must be 1 away from diffusions.

In digital design there are four kinds of contacts between metal1 and diffusion, one for each of the kinds of diffusion. Contacts between metal1 and ndiffusion are called ndcontact, ndiffcut or ndc. Contacts between metal1 and pdiffusion are called pdcontact, pdiffcut or pdc. Contacts between metal1 and nsubstratendiff are called nsubstratencontact or nncontact or nsc or nnc or nohmic. Lastly, contacts between metal1 and psubstratepdiff are called psubstratepcontact or ppcontact or psc or ppc or pohmic. All diffusion contacts must be at least 4 units wide, and must be separated from unrelated diffusion by 4 units, one unit more than the normal diffusion-diffusion separation. It is recommended that contacts to poly or diffusion be sized by multiple of 4. It is to warrant the cuts to be on the lambda grid.

Both poly and diffusion contacts appear on the screen as an overlap between the two constituent layers, with a cross over the contact area. All contacts must be rectangular in shape. Pdc may abut nsc and ndc may abut psc; all other contact abutments are illegal. Pcontact must be at least 2 units from any diffusion contact, even if the two contacts are electrically connected as on the top/right of the figure.

Transistors

P-type transistors are drawn as an area of poly over-lapping pdiffusion, with brown stripes in the transistor area. Magic accepts the names pfet or ptransistor for this layer. N-type transistors are drawn as an area of poly overlapping ndiffusion, with green stripes in the transistor area. The names nfet, and ntransistor may be used. Transistors of each type can be generated by painting polysilicon and diffusion on top of each other, or by painting the transistor layer explicitly. The design rules are the same for both types of transistor: transistors must be at least 2 units long and 3 units wide. Polysilicon used as gate must overhung transistor by at least x_{t1} units. Diffusion (ndiff or pdiff) must overhung transistor by at least x_{t2} units. They must be separated from nearby poly contacts by at least 1 unit. Polysilicon must be at least x_{t3} units away from diffusion, except where it is forming a transistor. Transistors must be at least 2 units away from each other.

Substrate contacts

There are several additional rules besides those in Section 2.6 that apply to substrate contacts. Substrate contacts are used to maintain proper substrate voltages and prevent latchup. Nncontact is used to supply a Vdd voltage level to the n-wells (or n-substrate) around p-transistors, and ppcontact is used to supply a GND voltage level to the p-wells (or p-substrate) around n-transistors. Nncontact must be separated from p-transistors by at least 3 units to ensure that the n+ implant doesn’t affect the transistor. Nncontact may be placed next to pdcontact in order to tie a transistor terminal to Vdd at the same time. Because of diode formation, nncontact

will not connect to adjoining pdiffusion unless there are contacts to metal1 to strap them together. Similar rules apply to ppcontact.

Spacings between P and N

Ndiffusion, ntransistor, ndcontact, and ppcontact must be kept far away from pdiffusion, ptransistor, pdcontact, and nncontact, in order to leave room for wells. Ndiffusion and pdiffusion must be 10 units apart. Substrate contacts can be 2 units closer to material of the opposite type (the wells needn't surround them by as many units), so nncontact need only be 8 units from ndiffusion, ppcontact need only be 8 units from pdiffusion, and nncontact and ppcontact need only be 6 units apart.

Wells and rings

For the most part, you should not need to draw explicit wells. However it is recommended that you use painting explicit well as a mean to plan for your layout topology. Nwell is generated around pdiffusion, ptransistor, pdcontact, and nncontact. Pwell is generated around ndiffusion, ntransistor, ndcontact, and ppcontact. Magic merges nearby well areas into single large wells when possible. For example, two ndiffusion or pdiffusion areas will share a single well if they are within 10 units of each other. There may be a few cases where you'd like to guarantee that certain areas are covered with wells, e.g. pads. For these cases you may paint the explicit layers nwell and pwell. Nwell appears on the screen as diagonal green stripes, and pwell appears as diagonal brown stripes. The explicit well layers that you paint will supplement the automatically-generated wells. Pwell must be at least five units from pdiffusion, ptransistor, or pdcontact, and three units from nncontact. Nwell must be at least five units from ndiffusion, ntransistor, or ndcontact, and three units from ppcontact. Nwell and pwell may abut but not overlap (if you paint one well on top of the other, the new one replaces the old one). All nwell and pwell areas that you paint must be at least 10 units wide and are separated from other wells of the same type by at least nine units.

Guard rings may be created using the nndiffusion and ppdiffusion layers. Nncontacts and ppcontacts should be used to strap the rings to Vdd and GND, respectively. Nndiffusion must be at least 3 units from pwell, 6 units from ppdifffusion or ppcontact, and 8 units from ndiffusion or ndcontact (these are the same rules as for nncontact). Similar rules apply to ppdiffusion. Guard rings are probably the only things that nndiffusion and ppdiffusion will be used for.

Overglass and pads

Normally, everything in the layout is covered by overglass in order to protect the circuitry. If you do not wish to have overglass in certain areas of the layout, there are two Magic layers you can use for this. The Magic layer pad should be used for drawing pads. It generates a hole in the overglass covering and also automatically includes metal1, metal2, and via. Pad is displayed as metal2 over metal1, with additional diagonal stripes. The rules for pads are in absolute microns, not lambda units: the pad layer must always be at least 100 microns on a side. Since this rule is in absolute units, it is not checked by the Magic design-rule checker. Pads will generally need to be modified in order to fabricate at different scale factors or for different flavors of CMOS.

An additional layer glass is provided to allow designers to make unusual glass cuts anywhere on the chip for probing or other purposes. This layer is drawn in dark diagonal stripes. Probe areas should generally be at least 75 microns wide, but Magic does not check this rule.