Magic Tutorial #10: The Interactive Router

Michael Arnold

O Division
Lawrence Livermore National Laboratory
Livermore, CA 94550

This tutorial corresponds to Magic version 6, and Irouter version 0.6.

Tutorials to read first:

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

Commands covered in this tutorial:

:iroute

Macros covered in this tutorial:

Cntl-R, Cntl-N

1. Introduction

The Magic interactive router, *Irouter*, provides an interactive interface to Magic's internal maze router. It is intended as an aid to manual routing. Routing is done one connection at a time, the user specifying a starting point and destination areas prior to each connection. The user determines the order in which signals are routed and how multi-point nets are decomposed into point-to-area connections. In addition parameters and special Magic *hint* layers permit the user to control the nature of the routes. Typically the user determines the overall path of a connection, and leaves the details of satisfying the design-rules, and detouring around or over minor obstacles, to the router.

The interactive router is not designed for fully automatic routing: interactions between nets are not considered, and net decomposition is not automatic. Thus netlists are generally not suitable input for the Irouter. However it can be convenient to obtain endpoint information from netlists. The *Net2ir* program uses netlist information to generate commands to the Irouter with appropriate endpoints for specified signals. Typically a user might setup parameters and hints to river-route a set of connections, and then generate Irouter commands with the appropriate endpoints via Net2ir. For details on Net2ir see the manual page *net2ir(1)*.

This tutorial provides detailed information on the use of the Irouter. On-line help, Irouter subcommands, Irouter parameters, and hint-layers are explained.

2. Getting Started - 'Cntl-R', 'Cntl-N', ':iroute' and ':iroute help'

To make a connection with the Irouter, place the cursor over one end of the desired connection (the *start-point*) and the box at the other end (the *destination-area*). Then type

Cntl-R

Note that the box must be big enough to allow the route to terminate entirely within it. A design-rule correct connection between the cursor and the box should appear. The macro

Cntl-R

and the long commands

:iroute
:iroute route

are all equivalent. They invoke the Irouter to connect the cursor with the interior of the box. Note that the last connection is always left selected. This allows further terminals to be connected to the route with the second Irouter macro, **Cntl-N**. Try typing

Cntl-N

A connection between the cursor and the previous route should appear. In general **Cntl-N** routes from the cursor to the selection.

There are a number of commands to set parameters and otherwise interact with the Irouter. These commands have the general form

:iroute *subcommand* [*arguments*]

For a list of subcommands and a short description of each, type

:iroute help

Usage information on a subcommand can be obtained by typing

:iroute help [*subcommand*]

As with Magic in general, unique abbreviations of subcommands and most of their arguments are permitted. Case is generally ignored.

3. :Undo and Cntl-C

As with other Magic commands, the results of **:iroute** can be undone with **:undo**, and if the Irouter is taking too long it can be interrupted with **Cntl-C**. This makes it easy to refine the results of the Irouter by trial and error. If you don't like the results of a route, undo it, tweak the Irouter parameters or hints you are using and try again. If the Irouter is taking too long, you can very likely speed things up by interrupting it, resetting performance related parameters, and trying again. The details of parameters and hints are described later in this document.

4. More about Making Connections - ':iroute route'

Start points for routes can be specified via the cursor, labels, or coordinates. Destination areas can be specified via the box, labels, coordinates or the selection. In addition start and destination layers can be specified explicitly. For the syntax of all these options type

:iroute help route

When a start point lies on top of existing geometry it is assumed that a connection to that material is desired. If this is not the case, the desired starting layer must be explicitly specified. When routing to the selection it is assumed that connection to the selected material is desired. By default, routes to the box may terminate on any active route layer. If you are having trouble connecting to a large region, it may be because the connection point or area is too far in the interior of the region. Try moving it toward the edge. (Alternately see the discussion of the *penetration* parameter in the wizard section below.)

5. Hints

Magic has three built-in layers for graphical control of the Irouter, **fence** (**f**), **magnet** (**mag**), and **rotate** (**r**). These layers can be painted and erased just like other Magic layers. The effect each has on the Irouter is described below.

5.1. The Fence Layer

The Irouter won't cross fence boundaries. Thus the fence layer is useful both for carving out routing-regions and for blocking routing in given areas. It is frequently useful to indicate the broad path of one or a series of routes with fence. In addition to guiding the route, the use of fences can greatly speed up the router by limiting the search.

5.2. The Magnet Layer

Magnets attract the route. They can be used to pull routes in a given direction, e.g., towards one edge of a channel. Over use of magnets can make routing slow. In particular magnets that are long and far away from the actual route can cause performance problems. (If you are having problems with magnets and performance, see also the discussion of the *penalty* parameter in the wizard section below.)

5.3. The Rotate Layer

The Irouter associates different weights with horizontal and vertical routes (see the layer-parameter section below). This is so that a preferred routing direction can be established for each layer. When two good route-layers are available (as in a two-layer-metal process) interference between routes can be minimized by assigning opposite preferred directions to the layers.

The rotate layer locally inverts the preferred directions. An example use of the rotate layer might involve an **L**-shaped bus. The natural preferred directions on one leg of the **L** are the opposite from the other, and thus one leg needs to be marked with the rotate layer.

6. Subcells

As with painting and other operations in Magic, the Irouter's output is written to the cell being edited. What the router sees, that is which features act as obstacles, is determined by the window the route is issued to (or other designated reference window - see the wizard section.) The contents of subcells expanded in the route window are visible to the Irouter, but it only sees the bounding boxes of unexpanded subcells. These bounding boxes appear on a special **SUBCELL** pseudo-layer. The spacing parameters to the **SUBCELL** layer determine exactly how the Irouter treats unexpanded subcells. (See the section on spacing parameters below.) By default, the spacings to the **SUBCELL** layer are large enough to guarantee that no design-rules will be violated, regardless of the contents of unexpanded subcells. Routes can be terminated at unexpanded subcells in the same fashion that connections to other pre-existing features are made.

7. Layer Parameters - ':iroute layers'

Route-layers, specified in the **mzrouter** section of the technology file, are the layers potentially available to the Irouter for routing. The **layer** subcommand gives access to parameters associated with these route-layers. Many of the parameters are weights for factors in the Irouter cost-function. The Irouter strives for the cheapest possible route. Thus the balance between the factors in the cost-function determines the character of the routes: which layers are used in which directions, and the number of contacts and jogs can be controlled in this way. But be careful! Changes in these parameters can also profoundly influence performance. Other parameters determine which of the route-layers are actually available for routing and the width of routes on each layer. It is a good idea to inactivate route-layers not being used anyway, as this speeds up routing.

The layers subcommand takes a variable number of arguments.

:iroute layers

prints a table with one row for each route-layer giving all parameter values.

:iroute layers *type*

prints all parameters associated with route-layer *type*.

:iroute layers *type parameter*

prints the value of *parameter* for layer *type*. If *type* is '*****', the value of *parameter* is printed for all layers.

:iroute layers *type parameter value*

sets *parameter* to *value* on layer *type*. If *type* is '*****', *parameter* is set to *value* on all layers.

:iroute layers *type* ***** *value1 value2 ... valuen*

sets a row in the parameter table.

:iroute layers * *parameter value1 ... valuen*

sets a column in the table.

There are six layer parameters.

active
Takes the value of **YES** (the default) or **NO**. Only active layers are used by the Irouter.

width
Width of routing created by the Irouter on the given layer. The default is the minimum width permitted by the design rules.

hcost
Cost per unit-length for horizontal segments on this layer.

vcost
Cost per unit-length for vertical segments.

jogcost
Cost per jog (transition from horizontal to vertical segment).

hintcost
Cost per unit-area between actual route and magnet segment.

8. Contact Parameters - ':iroute contacts'

The **contacts** subcommand gives access to a table of parameters for contact-types used in routing, one row of parameters per type. The syntax is identical to that of the **layers** subcommand described above, and parameters are printed and set in the same way.

There are three contact-parameters.

active
Takes the value of **YES** (the default) or **NO**. Only active contact types are used by the Irouter.

width
Diameter of contacts of this type created by the Irouter. The default is the minimum width permitted by the design-rules.

cost
Cost per contact charged by the Irouter cost-function.

9. Spacing Parameters - ':iroute spacing'

The spacing parameters specify minimum spacings between the route-types (route-layers and route-contacts) and arbitrary Magic types. These spacings are the design-rules used internally by the Irouter during routing. Default values are derived from the **drc** section of the technology file. These values can be overridden in the **mzrouter** section of the technology file. (See the *Magic Maintainers Manual on Technology Files* for details.) Spacings can be examined and changed at any time with the **spacing** subcommand. Spacing values can be **nil**, **0**, or positive integers. A value of **nil** means there is no spacing constraint between the route-layer and the given type. A value of **0** means the route-layer may not overlap the given type. If a positive value is specified, the Irouter will maintain the given spacing between new routing on the specified route-layer and pre-existing features of the specified type (except when connecting to the type at an end-point of the new route).

The **spacing** subcommand takes several forms.

:iroute spacing

prints spacings for all route-types. (Nil spacings are omitted.)

:irouter spacing *route-type*

prints spacings for *route-type*. (Nil spacings are omitted.)

:iroute spacing *route-type type*

prints the spacing between *route-type* and *type*.

:iroute spacing *route-type type value*

sets the spacing between *route-type* and *type* to *value*.

The spacings associated with each route-type are the ones that are observed when the Irouter places that route-type. To change the spacing between two route-types, two spacing parameters must be changed: the spacing to the first type when routing on the second, and the spacing to the second type when routing on the first.

Spacings to the **SUBCELL** pseudo-type give the minimum spacing between a route-type and unexpanded subcells. The **SUBCELL** spacing for a given route-layer defaults to the maximum spacing to the route-layer required by the design-rules (in the **drc** section of the technology file). This ensures that no design-rules will be violated regardless of the contents of the subcell. If subcell designs are constrained in a fashion that permits closer spacings to some layers, the **SUBCELL** spacings can be changed to take advantage of this.

10. Search Parameters ‘:search’

The Mzrouter search is windowed. Early in the search only partial paths near the start point are considered; as the search progresses the window is moved towards the goal. This prevents combinatorial explosion during the search, but still permits the exploration of alternatives at all stages. The **search** subcommand permits access to two parameters controlling the windowed search, **rate**, and **width**. The **rate** parameter determines how fast the window is shifted towards the goal, and the **width** parameter gives the width of the window. The units are comparable with those used in the cost parameters. If the router is taking too long to complete, try increasing **rate**. If the router is choosing poor routes, try decreasing **rate**. The window width should probably be at least twice the rate.

The subcommand has this form:

:iroute search [*parameter*] [*value*]

If *value* is omitted, the current value is printed, if *parameter* is omitted as well, both parameter values are printed.

11. Messages - ‘:iroute verbosity’

The number of messages printed by the Irouter is controlled by

:iroute verbosity *value*

If verbosity is set to **0**, only errors and warnings are printed. A value of **1** (the default) results in short messages. A value of **2** causes statistics to be printed.

12. Version - ‘:iroute version’

The subcommand

:iroute version

prints the Irouter version in use.

13. Saving and Restoring Parameters - ‘:iroute save’

The command

:iroute save ***file*****.ir**

saves away the current settings of all the Irouter parameters in file ***file*****.ir**. Parameters can be reset to these values at any time with the command

:source ***file*****.ir**

This feature can be used to setup parameter-sets appropriate to different routing contexts. Note that the extension **.ir** is recommended for Irouter parameter-files.

14. Wizard Parameters - ‘:iroute wizard’

Miscellaneous parameters that are probably not of interest to the casual user are accessed via the **wizard** subcommand. The parameters are as follows:

bloom
: Takes on a non-negative integer value. This controls the amount of compulsory searching from a focus, before the next focus is picked based on the cost-function and window position. In practice **1** (the default value) seems to be the best value. This parameter may be removed in the future.

boundsIncrement
: Takes on the value **AUTOMATIC** or a positive integer. Determines in what size chunks the layout is preprocessed for routing. This preprocessing (blockage generation) takes a significant fraction of the routing time, thus performance may well be improved by experimenting with this parameter.

estimate
: Takes on a boolean value. If **ON** (the default) an estimation plane is generated prior to each route that permits cost-to-completion estimates to factor in subcells and fence regions. This can be very important to efficient routing. Its rarely useful to turn estimation off.

expandDests
: Takes on a boolean value. If **ON** (not the default) destination areas are expanded to include all of any nodes they overlap. This is particularly useful if the Irouter is being invoked from a script, since it is difficult to determine optimal destination areas automatically.

penalty
: Takes on a rational value (default is 1024.0). It is not strictly true that the router

searches only within its window. Paths behind the window are also considered, but with cost penalized by the product of their distance to the window and the penalty factor. It was originally thought that small penalties might be desirable, but experience, so far, has shown that large penalties work better. In particular it is important that the ratio between the actual cost of a route and the initial estimate is less than the value of **penalty**, otherwise the search can explode (take practically forever). If you suspect this is happening, you can set **verbosity** to **2** to check, or just increase the value of **penalty**. In summary it appears that the value of penalty doesn't matter much as long as it is large (but not so large as to cause overflows). It will probably be removed in the future.

penetration

This parameter takes the value **AUTOMATIC** or a positive integer. It determines how far into a blocked area the router will penetrate to make a connection. Note however the router will in no case violate spacing constraints to nodes not involved in the route.

window

This parameter takes the value **COMMAND** (the default) or a window id (small integers). It determines the reference window for routes. The router sees the world as it appears in the reference window, e.g., it sees the contents of subcells expanded in the reference window. If **window** is set to **COMMAND** the reference window is the one that contained the cursor when the route was invoked. To set the reference window to a fixed window, place the cursor in that window and type:

:iroute wizard window .

15. References

[1] M.H. Arnold and W.S. Scott, ‘‘An Interactive Maze Router with Hints’’, *Proceedings of the 25th Design Automation Conference*, June 1988, pp. 672-676.