ROUTE Program Logic Manual

E. McCreight
Parc/CSL

Draft of June 2, 1978 11:17 AM

ROUTE is a Bcpl program that is part of the Parc/SDD/EOD electronic design automation system. Its function is to combine the net lists describing a number of logic drawings that together describe an entire logic board, and to generate a set of wiring orders sufficient to produce the board automatically.

It has been said that a university is an otherwise unrelated set of colleges sharing a common heating system. So it is with ROUTE. ROUTE is the current repository for a fairly large collection of unrelated functions sharing a common net list input format and wire list output format. These functions are such diverse things as automatic terminator assignment, wire length minimization, and wiring order determination.

ROUTE is further complicated by the necessity to carry out revisions incrementally. This means, among other things, that nets must be recognized as "the same" even if net names change, and that terminator assignment must change as little as possible.

0. Normal Operation

ROUTE is normally invoked from the Alto Executive with a command line like this:

> **Route**[/*switches*] *board*/**B** [*metric*/**M**] [*exhaustive*/**E**] [*heuristic*/**H**] [*boardlocation*/**L**] *file1 file2 file3 ...*

ROUTE normally reads a set of .nl-format files produced by the ANALYZE program and produces several output files. If the first .nl file in the input list (*file1*) had the name **AARGH.nl**, then the following output files would be produced

> *** AARGH.wl**, a file containing wiring orders for the stitchwelding program FAB to fabricate the board from scratch,
>
> *** AARGH.re**, a file containing error messages,
>
> *** AARGH.bp**, a file describing the external (backpanel, usually) connections of the board, and
>
> *** AARGH-*x*.nl**, a file for each each external connector type *x* describing the connections through that connector. These files are intended to be used as input for automatic backpanel routing.

If correction (revision) is specified, in addition ROUTE will read **AARGH.wl** and will produce two other output files:

> *** AARGH.wlnew**, instead of AARGH.wl. That’s so that if something goes awry, the original AARGH.wl is unchanged and the correction can be re-run. The other output file is

*** AARGH.ad**, a file containing FAB wiring orders to implement the revision on the pre-existing board.

The following switches have the indicated effect:

c: This specifies that correction is to be done.
b: This causes two .wl-format files to be produced, one containing all pins that are initially floating, and one containing all pins trace-wired to a power plane. This is useful for automated incoming board inspection.
m: This requests Multiwire-format wiring output, suitable for sending to Photocircuits, Inc.
h: This causes a list of hole positions to be produced in Multiwire-format. Photocircuits needs to know where not ___ to put the wires, too.
t: This causes a check list of trace-wired pins to be produced (not normally useful except for Board debugging (see below)).

The file *board* contains a concatenation of .BR files that collectively define all the Board routines (see below). The parameters *exhaustive* and *heuristic* together control the wire-routing part of ROUTE; their normal settings are 7 and 20. Better routings of long nets can be gotten at the expense of longer running times by increasing *heuristic*. The *metric* is either **Manhattan** (rectilinear) or **Euclidean** (as the crow flies); the default is **Manhattan**, which is faster. The *boardlocation* is a string that will be substituted for the connector name *x* in the **AARGH-*x*.nl** file mentioned above.

1. File Formats.

1.1 .NL format.

An .nl-format file is an ASCII text file that looks like this:

<comment line>
<IC line>
...
<IC line>
@
<net line>
...
<net line>

The <comment line> is ASCII text preceded by a ";" and terminated by a carriage return. ANALYZE generates the comment from a piece of "boiler plate" in the drawing. The standard boiler plate template is known as LogicBlock.sil.

The <IC line> has the following format:

<IC position>**:** <Short IC name>**(**<long IC name>/<npins>/<IC family>**);** <used group string>

For example,

h24: S04 (SN74S04/14/S) ; badfe

The <IC position> is a string in one of two forms: either a lower-case letter followed by a decimal number, or a string preceded by "#". In the former case, the name is normalized by suppressing leading 0's and then forcing the decimal number to be two digits long or longer. Thus **a004** and **a4** would both be normalized to **a04**, and **c04000** would be normalized to **c4000**. Interpretation of this normalized IC position string is strictly up to the board routines, but some guidelines have developed among board designers:

* It is pretty clear what ought to happen when the board socket and the IC are congruent. It is less clear what ought to happen when a Sip is plugged into a Dip socket, or an 8-pin Dip is plugged into a 16-pin socket, etc. Usually, something like **a41** means that pin 1 of the IC is supposed to go into pin 1 of the board's socket **a41**, and the rest of the pins of the IC plug into other pins of the board in an obvious manner defined by the board routines.

* For some boards, **a41** means that the IC is to be inserted in some board-standard part of **a41**. For example, D0 boards are covered with 20-pin sockets whose pin 10's are trace-wired to GND. The D0 board routines interpret the number **a41** on a 14-pin TTL IC as meaning that pin 1 of the IC goes in pin 4 of the socket, so that pin 7 of the IC goes in pin 10 of the socket, the GND pin.

* Most boards have adopted an offset convention. In this convention, **#3a41** means that pin 1 of the IC is to be offset 0.3" in the direction of increasing socket pin numbers from pin 1 of the socket. For the D0 board above, **a41** and **#3a41** would specify the same position for a 14-pin TTL IC. If you wanted to put an 8-pin Sip in pins 12-19 of a 20-pin 300-mil-wide Dip socket, you would say **#1_3a41**, signifying a "sideways" shift of 300 mils and a "vertical" shift of 100 mils.

* More positioning conventions will likely evolve. The idea is for any reasonable positioning to be possible, and for common ones to be easily specified, preferably to happen by default.

The <IC family> is a string from which ROUTE infers a number of characteristics of the IC, such as

a) to what pins (if any) automatic terminator assignment applies,
b) what fixed voltages are applied to what pins,
c) how to compute the co-ordinates of pin *i* given the co-ordinates of pin 1.

The <net line> has the following format:

<net name>**:** <pin>**,** <pin>**, ...,** <pin>

For example,

Sin.15: j26.13o, E146
stor08.sil+8: g22.2i, k25.12o, j23.11i
WEo': g22.3i

A <net name> is an alphanumeric string beginning with an alphabetic character and optionally ending with the character "!". Two net names differing only in the final "!" are considered to be the same for matching purposes, and the "!" affects only automatic terminator assignment.

In general, a <pin> may be one of the following:

<Connector string><decimal number><**i**/**o**/**p**/nothing>
<IC position string>**.**<decimal number><**i**/**o**/**p**/nothing>

A connector string is something like **E** or **C**. It may not end with a period or digit. An IC position string is something like **a41**. It can contain periods but may not end with one. Interpretation of the connector strings and IC position strings rests with Board routines to be described later. The final letter **i** or **o** or **p** indicates whether ANALYZE believes that the pin is an input pin, an output pin, or a power pin. These beliefs form the basis of some of ROUTE's warning messages.

1.2 .wl *format.*

An .wl-format file is an ASCII text file that looks like this:

<board type line>
<comment line>
...
<comment line>
<IC line>
...
<IC line>
@
<basic wiring command>
...
<basic wiring command>

The <board type line> consists of a string followed by a carriage return. The string uniquely encodes the board type. The <comment line>'s are just the comment lines collected from the various .nl files. An <IC line> now has the following form:

<IC position>**: (**<long IC name>/<npins>/<IC family>**);** <pin number>**,...,**<pin number>

where the <pin number>'s are pins unused by any of the input .nl files. For example,

i26: S04 (SN74S04/14/S); 8,13,14,15,16

A <basic wiring command> looks like one of the following:

<CR>**CALIBRATE: <**<decimal command number>**>;** <operator instructions><CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

or

<CR>**DISCONNECT: <**<decimal command number>**>**<CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

or

<CR><net name>**: <<**decimal command number>**> (**<decimal wire length>**)**<CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

A <co-ordinate> is a string of the form:

{<decimal number>**,**<decimal number>**}**

where the two decimal numbers are interpreted as distances in the x and y axes, measured in units of .025 inch, from the "origin" of the board (an arbitrary fixed position).

For example,

Sout.00: <8> (7)
b01.05i {052,007} C176 {052, 000}

CALIBRATE is a command that causes FAB to solicit operator assistance in attaching the board to the x-y table, attaching a working tool, and locating four calibrating points on the board, which must form the corners of a rectangle. After that FAB can find every point by itself until the board is removed from the table.

DISCONNECT is a command to operate a milling tool to isolate a stitchweld pin from the trace-wired net to which it was originally connected during board manufacture. This facilitates installing TTL IC's in sockets intended for ECL IC's, for example.

<net name> is an implicit command to wire up the named net in the same order as the pins are mentioned.

1.3 .ad *format.*

An .ad-format file is almost a superset of a .wl-format file. There are two differences. First, an .ad-format file does not have the <board type line>. Second, the .ad-format file allows several additional wiring commands:

<CR>**UNPLUG: <<**decimal command number>**>**<CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

or

<CR>**DISCARD: <<**decimal command number>**>**<CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

or

<CR>**RECONNECT: <<**decimal command number>**>**<CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>

...

or

<CR>**DELETE: <<**decimal command number>**>;** <net name><CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

UNPLUG and **DISCARD** are commands designed to give the stitchwelding machine unrestricted access to stitchweld pins on which it will need to work. The difference is that **DISCARD** means that the IC should not be replaced after the update, while **UNPLUG** means that the IC will be re-plugged afterwards. **UNPLUG** is actually a list of positions that will be re-plugged after the change, so it may contain board positions that originally were empty. Sorry for the confusion.

RECONNECT is a command to restore the connection from trace to pin that was earlier destroyed by a **DISCONNECT** command. This would normally be done by soldering. The Board routines specify under what conditions the operator can be asked to do this.

DELETE is a command to remove a net that was wired onto the board in an earlier revision.

1.4 Multiwire Net format

1.5 Multiwire Hole format

2. MetaProgram.

First, let us imagine that ROUTE is being invoked to do its simplest task: collect together several .nl files and produce a .wl file, a .bp file, some -*x*.nl files, and a .re file. Processing proceeds as follows:

1. The command line is processed to extract parameters. These include:

    a. the names of the .nl files,
    b. the names of the .wl, .bp, and .re files,
    c. the name of a .br-format file (or concatenation of .br-format files) containing compiled Bcpl Board routines,
    d. whether or not re-work is desired (/R), and
    e. parameters to control wire length minimization.

2. Read in each .nl file. For each <net name>, accumulate a list of all <pin>'s contained in that net.

3. See whether any of the pin assignments in (2) above conflict with so-called trace-wired nets; that is, board-defined nets that are wired by PC-board traces. If so, and if this is permissible for the board type, disconnect the offending pins from their trace-wired nets.

4. All pins connected to trace-wired nets are partitioned into clusters according to the closest pins that are still trace-wired. These clusters are then reassigned to new

nets with names like **VCC1**, **VDD35**, etc.

5. All nets are routed if this has been requested. For each net this involves a permutation of the net order so as to minimize the total wire length. Additional termination pins may be added to nets in this step.

6. The nets are then sorted into an order intended to optimize the wiring process. For stitchwelding this is currently believed to be:

a) nets with very short arcs first, so other wires will not interfere with them,

b) if two nets have shortest arcs of the same length, wire the shorter net first.

7. All the output files are created from the data structures built up in steps 1-6.

2.1 Automatic terminator assignment.

The ECL logic family does not work properly unless each net contains at least one terminating resistor. Assignment of terminating resistors to nets by hand is a tiresome task, and it is reasonable that the DA system should do it. Particularly for long wires, the driver should be on one end of a wire and a terminating resistor be on the other, or else the driver should be in the middle and terminators on both ends. ROUTE is the only program with enough information about board position and wire routing to be able to do rational terminator assignment.

First, let us expand on step 5 above to explain how termination comes about:

5a. A permutation of the net is chosen to minimize the wire length, subject to the following constraints:

i) the first edge (or cable) pin is constrained to lie at the end of the permutation, or

ii) if there is any ECL output pin in the net, and if exactly one pin in the net is marked as an output pin, and if the net contains no edge or cable pins, then if the resulting net would be no longer than 1.2 times the length of the constrained net, the output pin is constrained to lie at the end of the permutation.

5b. If no instance of the net name ended in the character "!" (signifying that termination is to be ignored) and there is an ECL output pin in the net, and there are no terminating resistors explicitly included in the net, then either one or two terminating resistors will be assigned to the net. Two resistors will be assigned if the net is longer than 4 inches, or if the net contains more than one output pin, or if any output pin is not at the end of the net. One resistor will be assigned otherwise.

5c. If any terminating resistors are to be assigned, they are assigned either at the ends of the nets or between the next-to-end and end pins in the net so as to minimize the increase in wire length (except if an unterminated stub longer than 3 inches would result). Terminators are chosen as close as possible to the end pins. If only one terminator is assigned, it is assigned on the opposite end of the net from the output or edge pin.

2.2 Correction.

Another practical consideration is correction (revision). It should be possible to correct a set of drawings and have the size of the resulting wiring change relate to the size of the change in the drawings. This is done by reading the previous .wl file and only changing the wiring of a net if it differs between the old .wl file and the new one. Unfortunately, one of the least significant physical features of a net is its name, so ROUTE must be able to recognize identical nets as identical even if their names change. If re-work is requested, a new step 4.5 is inserted after the nets are completely specified but before they are routed:

> 4.5 For each net in the old .wl file, determine what new net it is the same as, except for terminating resistors. If it is not the same as any new net, then mark it for deletion in the .ad file. If it is the same as some new net, and if the termination for the old net makes sense for the new net, and if the new net has no explicit termination of its own, then route and terminate the new net exactly as the old one was, and mark it as routed and terminated so that it will not be worked on in step 5 nor output to the .ad file.

3. Internal Data Structures.

3.1 Names and Namees.

The basic organizing concept in ROUTE is the "name". Nets have names, IC types have names, board positions have names, etc. ROUTE maintains a single name data structure, and attached to each name is a list of named objects with that name. The name data structure is a hash table where each bucket is a pointer to a list of **name** blocks that all hash to that bucket. A **name** block looks like this:

```
structure name:
    [ next word         // next name block in bucket list, or nil
    mark word    // =-1. Namee list is circular and ends here.
    nameString @string
    ]
```

The **name** block is immediately followed by a namee block, several types of which are described below.

3.1.1 Nets.

One namee is a net. A **net** block looks like this:

```
sturcture net:
    [ next word // to next namee with this name
    flags bit 4
    unused bit 8
    type bit 4      // =net
    pinList word        // pointer to first pin of pin list
    shortestarc word = netnum word = minSperge word
    netlength word
    ]
```

3.1.2 IC instances.

Another namee is an IC instance. The name denotes the board position. An **icinst** block looks like this:

```
sturcture icinst:
    [ next word // to next namee with this name
    type word     // =icinst
    ictype word         // pointer to ictype block for this IC
    pinattribute word
    pin^1,npins         // each one links to the next pin in
        // the pin list of its net, or nil
    ]
```

Note that one can chain one's way into an **icinst** block at some undetermined index in its pin vector, and then get properly aligned with the **icinst** block by scanning backward until detecting type=icinst, which is an illegal value for pin, pinattribute, and ictype words.

3.1.3 IC types.

Another namee is the IC type. An **ictype** block looks like this:

```
sturcture ictype:
    [ next word // to next namee with this name
    npins bit 12 // same as npins in icinst
    type bit 4      // =ictype
    icclass word        // pointer to IC class containing this type
    outpins^1,npins bit 1         // one bit per pin, true if pin
        // ever used in any IC instance as output
    ]
```

3.1.4 IC classes.

The final namee is the IC class. An **icclass** block looks like this:

```
structure icclass:
    [ next word // to next namee with this name
    isTraceWired bit 1            // needs termination?
    isConnector bit 1 // is this a terminator IC?
    printUsedList bit 1
    unused bit 9
    type bit 4      // =icclass
    PinOffset word
        // PinOffset(npins, pinNo, lv XOffset, lv YOffset) is
        // a routine that computes the X- and Y- offsets
        // of the given pin from pin 1, in 25-mil units.
        // For a standard 14-pin DIP, pin 1 would result in
        //     {0,0}, pin 2 would result in {4,0}, and pin 14 would
        // result in {12,0}.
    PinAttributes word
        // PinAttributes(icinst, pinNo) = attributes, such as
        // isEcl or isTerminator
    ImplicitICNets word
```

```
        // ImplicitICNets(npins, icInstNameString) is
        // a routine that writes
        // nets for such things as IC power and ground onto
        // a file called "implicit.nl"
    npins word
        // # pins, overridden by ictype block
    ... and other fairly esoteric stuff for terminators and trace-wired nets
    ]
```

3.2 Boards.

A board is a set of Bcpl subroutines dynamically loaded into ROUTE. It is represented as one or more concatenated .BR files. These subroutines are:

FindIndexFromCoord(xPos, yPos, picclass, pPinNo) = index

> This routine finds an integer from 1 to maxPins that uniquely represents the board pin at co-ordinates xPos, yPos. It returns 0 if there is no board pin at co-ordinate xPos, yPos. If xPos = yPos = -1, then it returns maxPins+1. In addition, if the board pin is initially connected to some trace-wired net, @picclass is set to the icclass describing that trace-wired net and @pPinNo is set to the number of the pin within the trace-wired net.

DeclareInitialNets(TWBuild, TermBuild, ConnBuild)

> This routine declares all the trace-wired nets, terminator sets, and connectors that the board has.

ZeroTablePoint(point) = string

> If point=0 then the string name of the board is returned. If point is from 1 to 4 the string name of the proper calibration point is returned. The calibration points should form a rectangle on the board.

LevelTransform(level, x, y, px, py, pName, pPull, pWire) = exists

> A given pin is described by different co-ordinates according to whether a board is positioned wiring-side up or component-side up. For each value of level, this subroutine implements a transformation from internal co-ordinates x,y to printing co-ordinates @px,@py. @pName is set to a string giving the name of this level. @pPull is set according to whether components can be removed at this level. @pWire is set according to whether wiring can be done at this level.

FindCoordFromString(s, px, py, vop1, hop1) = {absolute, relative, illegal}

> This routine takes the string name of a board location and computes the internal co-ordinates x,y of that position. It then adds vop1 and hop1 as the offsets vertically (the long way) and horizontally (the short way) of some desired pin assuming that pin 1 is installed at x,y. The result is placed in @px,@py. The result is absolute if the pin is legal, and illegal otherwise.

BoardPinCoord(s, icinst, pinNo, px, py) = {true, false}

This routine is normally defined by ROUTE itself (using the board's FindCoordFromString), but the board module can override ROUTE's definition. This routine takes the string name of a board location, a pointer to an IC instance, and a pin number, and computes the internal co-ordinates x,y of that pin. The result is true if the pin position is legal, and false otherwise. This routine would be defined instead of FindCoordFromString if you want to do screwball ball things like put TTL IC's upside-down in Ecl-wired sockets.