M4 Macros for Electric Circuit Diagrams in LaTeX Documents

J. D. Aplevich

Version 4.1

1 Introduction

This document describes a set of macros, written in the **m4** macro language[4], for producing electric circuits and other diagrams in LaTeX documents. The macros evaluate to drawing commands in **pic**, a drawing language[5] which is very simple to learn. The result is a system with the advantages and disadvantages of TeX itself, which is macro-based, non-wysiwyg, and which uses ordinary character input.

A collection of basic components and conventions for their internal structure are described. For particular drawings it is often convenient to create new macros, or combinations of them, using consistent conventions, so macros such as these are only a starting point. The IEEE standard[1] has been followed. The macros described here make extensive use of the features of **pic** and have been designed, where possible, to be an extension of the language.

2 Macro usage

The diagram source file is preprocessed as illustrated in Fig. 1. The source, together with the pre-defined macros, is first passed through **m4**, and then through a **pic** interpreter which produces a `.tex` file to be inserted into the `.tex` source using the `\include` command.

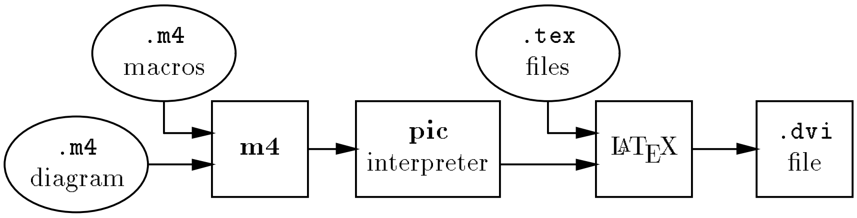

Figure 1: Inclusion of figures and macros in the LaTeX document

The file, `cct.m4,` say, is processed as shown:

```
m4 $HOME/lib/libcct.m4 cct.m4 > cct.pic
```

after which the `.pic` file must be converted to a `.tex` file containing, for example, **tpic** specials, LaTeX graphics, (or other graphics commands, such as for **mfpic**[6] or **PSTricks**[8]) which LaTeX will convert or include in a `.dvi` file. The `.dvi` file is then to be viewed or printed by a driver capable of interpreting any `\special` commands inserted by the **pic** processor. In the above example, `$HOME/lib` is the directory in which files `gpic.m4` and `libcct.m4` have been installed.

3 The Pic processor

The **pic** language defines the following elementary graphics objects: `line, box, arc, arrow, ellipse, circle, spline,` text string, and blocks containing simpler objects. In addition, local and global variables, elementary mathematical functions, structured objects, looping and branching, and elementary macro and file facilities are available. The graphics objects are drawn by absolute location, or relative to previously-defined objects or locations. At all times a current location and drawing direction are defined.

To convert **pic** to TeX input one can use[3] **gpic -t** with a printer driver that understands **tpic** specials, typically[7] **dvips**. Thus, for the previous example the command

```
gpic -t cct.pic > cct.tex
```

creates a file for inclusion in the LaTeX document. **Pic** processors contain basic macro facilities, so some of the concepts applied here require only a **pic** processor.

With judicious use of macros the features of both **m4** and **pic** can be exploited. (The fastidious might observe that there are 3 languages being scrambled: **m4**, **pic**, and the **tpic, tex** or other output, not to mention the meta-language of the macros, and that this mixture might be a problem, but experience implies otherwise.)

A **pic** manual[5] can be obtained by emailing to `netlib@research.att.com` with the message body `send 116 from research/cstr,` or by ftp as

`ftp://netlib.att.com/netlib/att/cs/cstr/116.ps.Z.`

Further explanation is available, for example, from the **gpic** 'man' page, part of the GNU **groff** package. For a discussion of the use of "little languages" in document production, and of pic in particular, see Chapter 9 of [2].

4 Basic two-terminal elements

All fundamental two-terminal element macros have been constructed according to the conventions described below.

First, all drawing macros have default arguments, so that only arguments different from default need be specified. The arguments are given in Sec. 14.

Consider the resistor shown in Fig. 2, which also serves as a useful example of **pic** commands.

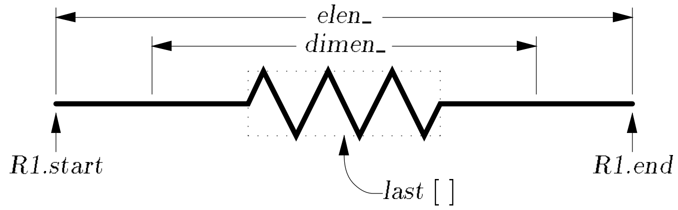

Figure 2: Resistor (with label $R1$), showing the enclosing block

The first line of a source file begins with `.PS` with optional further arguments, and the last line is normally `.PE`. The first part of the source for Fig. 2 is shown below:

```
.PS
   cct_init
   linewid = 2.0
   linethick_(2.0)

R1: resistor
```

These lines and the remaining source lines are explained below:

1. The first line invokes an almost-empty macro which initializes local variables used by some circuit-element macros. This macro can be customized to set line thicknesses, maximum page sizes, scale parameters, or other global quantities as desired.

2. By default, the size of two-terminal elements is a multiple of the **pic** variable `linewid,` which has initial value 0.5 in., so the value 2.0 is assigned to this variable to make a suitably big diagram for this manual. The body size of an element is determined by the macro `dimen_,` which evaluates by default to `linewid,` and the default element length length is `elen_,` which is `dimen_*3/2` by default. These dimensions can all be customized if necessary. (For resistors, the length of the body is `dimen_/2,` and the width is `dimen_/6.`)

3. The macro `linethick_` sets the thickness of subsequent lines (to 2.0pt in the example).

4. The sequence of drawing commands to which a two-terminal macro is expanded begins with the command '`line invis` ⋯' where the part following '`line invis`' is the first argument of the macro if it exists, otherwise this part specifies length `elen_` in the current direction. The invisible line is first drawn, and the element is drawn on top of the line. The current direction is initially to the right, so in the example the resistor is drawn to the right.

5. The element—rather the initially-drawn invisible line—can be given a name (`R1` in the example, so that points `R1.start` and `R1.end` are defined as shown, also `R1.center` is defined). Pic place names begin with an upper-case letter, whereas variable names begin with a lower-case letter.

6. The element body is enclosed by a block, which later can be referenced for placing labels around the element. The block corresponds to an invisible rectangle of which the top and bottom are horizontal, and the sides of which are vertical, regardless of the direction in which the element is drawn. In the diagram a dotted box has been drawn to show the block boundaries.

7. The last sub-element, identical to the first, in each two-terminal element is an invisible line that can also be referenced later to place labels or other elements.

To produce Fig. 2 the following embellishments were included after the previously-shown source:

```
   thinlines_
   box dotted wid last [].wid ht last [].ht at last []

   spline <- down 0.2 then down 0.1 right 0.1 then right 0.1 \
      from last [].s
   "{\sl last} [ ]" ljust

   arrow <- down 0.2 from R1.start+(0,-0.05); "{\sl R1.start}" below
   arrow <- down 0.2 from R1.end+(0,-0.05); "{\sl R1.end}" below

   dimension_(from R1.start to R1.end,0.45,{\sl elen\_},0.4)
   dimension_(right_ dimen_ from R1.c-(dimen_/2,0),0.3,
      {\sl dimen\_},0.5)
.PE
```

These lines have the following explanation.

- The line thickness is set to the default thin value of 0.4pt, and the box displaying the element body block is drawn. Notice how the width and height can be specified, and the box centre positioned at the centre of the block.

- The next paragraph draws two objects, a spline with an arrowhead, and a string left justified at the end of the spline. Other string-positioning modifiers than `ljust` are `rjust, above,` and `below.` Lines to be read by **pic** can be continued by putting a backslash as the rightmost character. Where meaningful, `Pic` objects have pre-defined compass corners, including `.s` in the example, others being `.n, .e, .w, .ne, .nw, .sw, .se, .c, .start, .end.` Furthermore any position, such as `R1.c` in the example, has an x-coordinate and y-coordinate usable in expressions as, for example, `R1.c.x` and `R1.c.y.`

- The next paragraph draws the arrows and labels at the ends of the resistor. Positions on the drawing can be defined in many ways, including referring to a corner of an object or object-name, or by adding paired arithmetic expressions to a position as illustrated.
- The last paragraph invokes a macro for dimensioning diagrams. Lines can be broken before a macro argument because **m4** ignores white space before arguments.

5 Macro arguments

Fig. 3 is a table of the 2-terminal elements included with this package, all drawn with the current direction to the right, which is the default pic drawing direction. Some elements are included more than once to illustrate their arguments. In the **m4** language, macro arguments are included

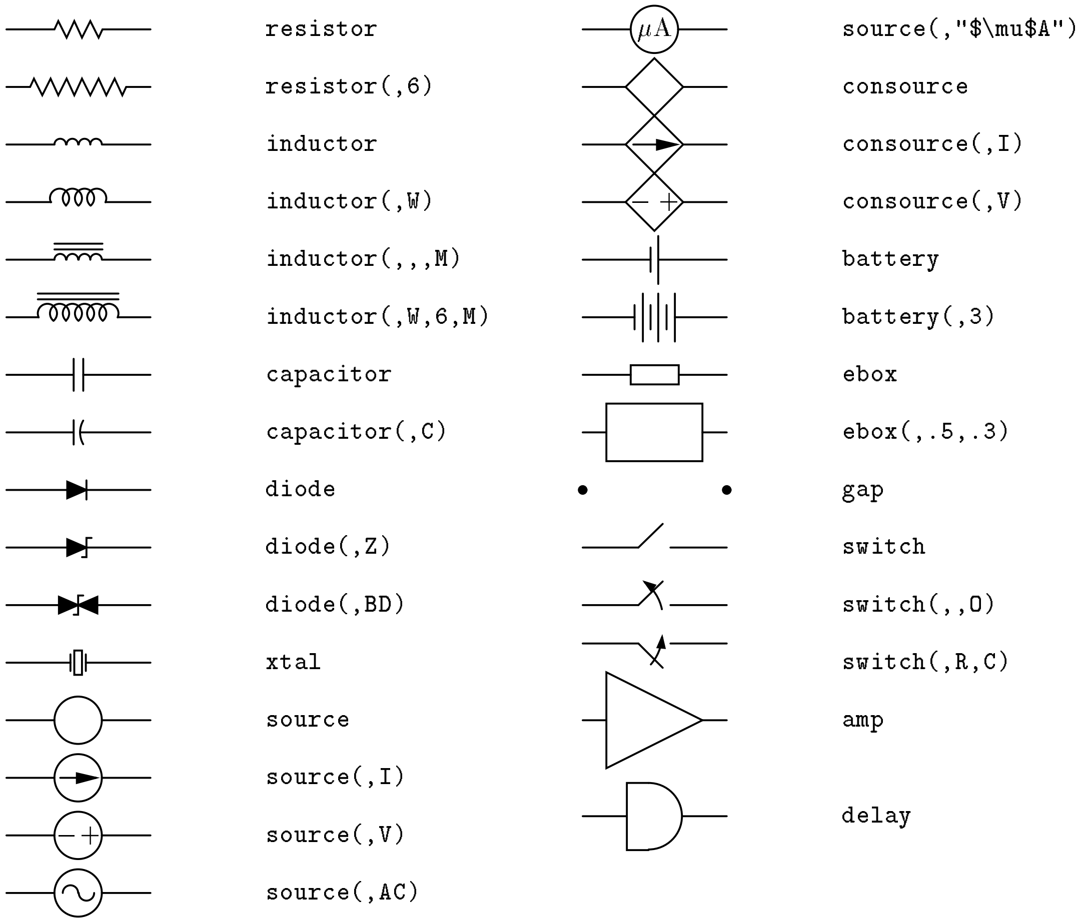

Figure 3: 2-terminal element macros, with some variations

within parentheses which follow the macro name, without spaces between the name and the opening parenthesis.

The first argument of the two-terminal elements, if included, defines the line direction and length along which the element is drawn. Other arguments of several macros are used to produce variants of the default elements. Thus, for example,

`resistor(up_ 1.25)`

draws a resistor 1.25 units long up from the current position. The name `up_` is a macro that resets the current macro and `pic` directional parameters to point up.

Instead of defining a direction and distance as above, some variants of the first argument are

`up_ 1.25 from` *position*
`to` *position*
`from` *position* `to` *position*
`with` *corner* `at` *position*
where for a line, the *corner* is one of `.start`, `.end`, or `.center`.

A position may be absolute, such as *(expression, expression)*, or *(position,position)*, where the latter takes the x-coordinate from the first position, and the y-coordinate from the second. Positions can be named, thus

`Node1: (0,0); Node2: (1,1); inductor from Node1 to Node2,`

or, for example,

`Amp1: opamp with .In1 at Here; line up 0.5 from Amp1.Out`

where `Amp1` becomes the name for a block containing positions `In1` and `Out` predefined in the opamp macro. Referring to `Amp1` is the same as `Amp1.center,` a compass corner can be used, e.g. `Amp1.n` for the top, and `Amp1.In1` is the named position within the block.

The name `Here` is the default for the current drawing position.

Another possibility for defining positions is

expression `of the way between` *position* `and` *position*

which is abbreviated as

expression < *position* , *position* >

but care has to be used in giving this construction to `m4`, since the comma may have to be put within quotes, `` `,' `` to distinguish it from the `m4` argument-separator.

Positions can refer to previously-drawn elementary **Pic** objects within the current block, e.g.

`resistor from 1st line.start to 3rd last [].s,`

and finally, positions can be calculated using variables, which must begin with a lower-case letter, and of which the scope is the current block. Thus, for example,

```
theta = atan2(B.y-A.y,B.x-A.x)
line to Here+(3*cos(theta),3*sin(theta)).
```

A `Pic` manual should be consulted for more examples, and for other facilities, such as the branching facility,

`if` *expression* `then` { *anything* } `else` { *anything* },

the looping facility,

`for` *variable* = *expression* `to` *expression* `by` *expression* `do` { *anything* },

operating-system commands, string handling, and file inclusion.

6 Branch-current arrows

The macro

```
b_current(label,position,direction)
```

draws an arrow on the last-drawn two-terminal element (i.e. line), inward by default, outward if the third argument is non-blank. The first argument is an optional label, of which the default position is the macro `above_`, which evaluates to above if the current direction is right, or to ljust, below, rjust if the current direction is respectively down, left, up. A non-blank second argument specifies another relative position, for example `below_`, which places the label below with respect to the current direction, or an absolute position, for example `below,` or `ljust,` with respect to the arrowhead.

7 Labels

Macros for labelling two-terminal elements are included:

```
resistor(right_ 1.5)        resistor(left_ 1.5)
b_current(i)                b_current(i,below_)
```

```
resistor(left_ 1.5)         resistor(right_ 1.5)
b_current(i)                b_current(i,below)
```

```
resistor(right_ 1.5)        resistor(left_ 1.5)
b_current(i,below_)         b_current(i,below)
```

Figure 4: Illustrating `b_current`

```
llabel(arg1,arg2,arg3)
clabel(arg1,arg2,arg3)
rlabel(arg1,arg2,arg3)
dlabel(long,lat,arg1,arg2,arg3)
```

The first places the three arguments, which are treated as math-mode strings, on the left side of the element block *with respect to the current direction: up, down, left, right.* The second places the arguments along the centre, and the third along the right side. Thus a simple circuit example with labels is

```
.PS
cct_init
define(`dimen_',0.75)
loopwid = 1; loopht = 0.75
  source(up_ loopht); llabel(-,v_s,+)
  resistor(right_ loopwid); llabel(,R,); b_current(i)
  inductor(down_ loopht,W); rlabel(,L,)
  capacitor(left_ loopwid,C); llabel(+,v_C,-); rlabel(,C,)
.PE
```

which produces Fig. 5. The macro `dlabel` performs the above functions for an obliquely-drawn ele-

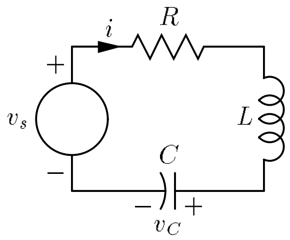

Figure 5: A loop with labelled elements

ment, placing the three macro arguments at `vec_(-long,lat)`, `vec_(0,lat)`, and `vec_(long,lat)` respectively relative to the centre of the element.

8 Diagram scaling

The finished size of a diagram or of elements whithin it can be easily adjusted.

First, the line, e.g. '.PS 4.5' scales the entire drawing to a width of 4.5in, no matter what absolute lengths are in the interior.

Second, resetting the pic variable 'scale', e.g. 'scale = 2.54' divides all lengths by 2.54, as one would to use cm instead of inches.

Third, the two-terminal elements all have default length and body size which are multiples of the pic variable 'linewid,' so setting this variable resets default element dimensions. The scope of a pic variable is within the current block, so a construct such as

```
linewid = 0.75
   ... drawing commands
B: [linewid = 1.25
      ... drawing commands
      ]
   ... more drawing commands
```

changes the size of elements within block `B`. Alternatively, it is permissible to redefine the default length `elen_` or default body size paramenter `dimen_`.

Text strings do not scale at all unless adjusted separately.

9 Other circuit elements

Some basic elements are not two-terminal. Fig. 6 shows a ground with and without a stem, an operational amplifier, and a simple transformer. By convention, such elements are enclosed in a block, and thus can be named and can contain named locations in the interior, such as E, B, C for a bipolar transistor. An invisible locating line can be optionally specified by the first argument, as for the two-terminal elements. Instead of positioning by the first line, the enclosing block may be positioned, thus, *element*`(at` *position*`)`, or using its compass corners, thus, *element* `with` *corner* `at` *position*, or, when the block contains a pre-defined location, using *element* `with` *location* `at` *position*. Macro arguments are given for each element in Sec. 14.

The operational amplifier is enclosed in a block containing the three pre-defined internal locations shown on the diagram, `In1`, `In2`, and `Out`, which can be referenced in later commands, for example as '`last [].Out`.' The first argument defines the direction and length of the opamp, but the position is determined either by pre-defined enclosing block of the opamp, or by a construction such as '`opamp with .In1 at Here`' which places the internal position `In1` at the specified location. There are optional second and third arguments for which the defaults are `$-$` and `$+$` respectively.

The transformer has pre-defined internal locations `P1`, `P2`, `S1`, and `S2`. The first argument specifies the direction and distance from `P1` to `P2`, with position determined by the enclosing block as for opamps. The second argument places the secondary side of the transformer to the left or right of the drawing direction.

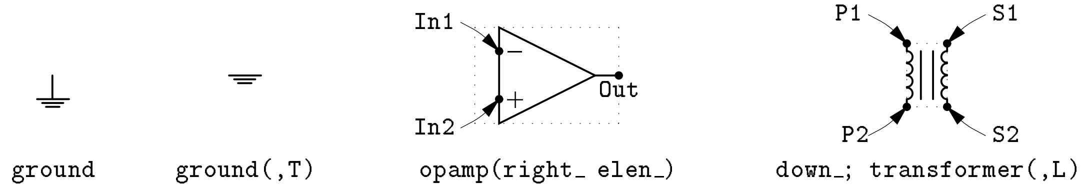

Figure 6: Miscellaneous basic elements

Fig. 7 shows the variations of bipolar transistor macro `bi_tr`, which contains pre-defined internal locations *E*, *B*, *C*. The first argument defines the distance and direction from *E* to *C*, with location determined by the enclosing block as for other elements, and the base placed to the left or right of the current drawing direction according to the second argument. Setting the third argument to '`P`' creates a PNP device instead of NPN, and setting the fourth to '`E`' draws an envelope around

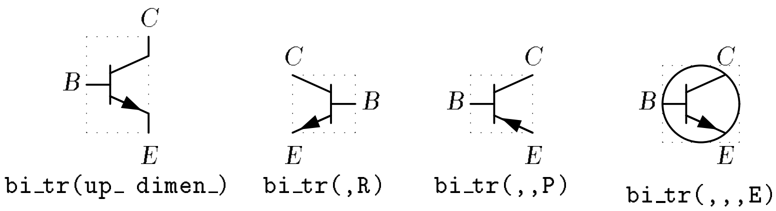

Figure 7: Bipolar transistor variants (current direction upward)

the device. Some FETs with pre-defined internal locations *S*, *D*, and *G* are also included, with the same arguments as `bi_tr`, as shown in Fig. 8. The number of possible semiconductor symbols is very large, so these macros must be regarded as prototypes.

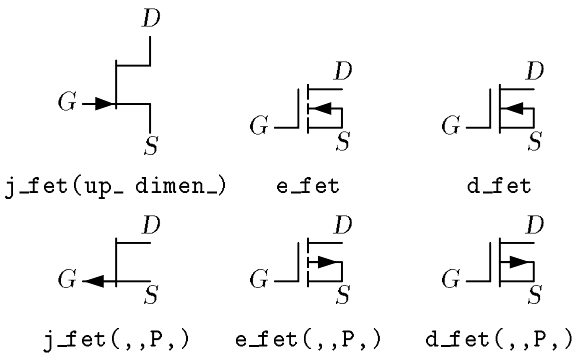

Figure 8: JFET, insulated-gate enhancement and depletion MOSFETS

Some other non-two-terminal macros are `dot` which has an optional argument ‘`at` *location*’, the line-thickness macros, the fill macro, and `hop`, which is useful to show non-touching cross-overs of conductors.

10 Directions

Aside from its block-structure capabilities, looping, and macros, **pic** has a very useful concept of the current point and current direction, the latter unfortunately limited to `up, down, left, right.` Objects can be drawn at absolute locations or placed relative to previously-drawn objects. These `.m4` macros need to know the current direction so whenever `up, down, left, right` are used they should be written as `up_, down_, left_, right_` which are macros.

To draw circuit objects in other than the standard four directions, the macros `Point_(`*`degrees`*`)`, `point_(`*`radians`*`)`, and `rpoint_(`*`rel linespec`*`)` re-define the direction-cosine macros `x_, y_` which are used within element-drawing macros. Thus ‘`Point_(-30); resistor`’ draws a resistor along a line with slope -30 degrees. ‘`rpoint_(to Z)`’ sets the current direction cosines to point to location Z. Macro `vec_(x,y)` evaluates to the position `(x,y)` rotated by the argument of the previous `Point_`, `point_` or `rpoint_` command. The macro `rvec_(x,y)` evaluates to position `Here + vec_(x,y)` and is the principal device used to define relative locations in the circuit macros. Thus, ‘`line to rvec_(x,0)`’ draws a line of length `x` in the current direction.

Fig. 9 shows a circuit drawn using the above macros. The source for the figure is below, and illustrates that some hand-placement of labels using `dlabel` may be useful when elements are drawn other than horizontal or vertically.

```
.PS
thicklines_
define(‘elen_’,‘dimen_’)
define(‘sourcerad_’,‘dimen_*0.2’)
```

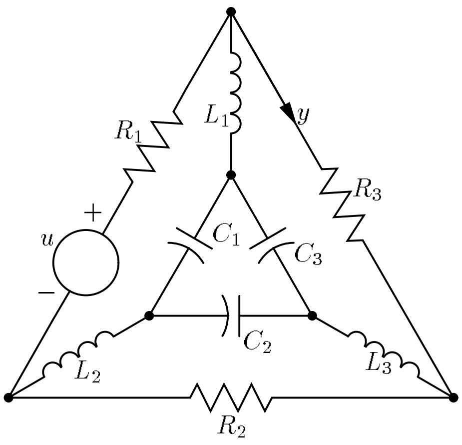

Figure 9: Illustrating elements drawn at oblique angles

```
linewid = 0.85

Ct:dot
   Point_(-60); capacitor(,C); dlabel(0.14,0.14,,,C_3)
Cr:dot
   left_; capacitor(,C); dlabel(0.14,0.14,C_2,,)
Cl:dot
   down_; capacitor(from Ct to Cl,C); dlabel(0.14,0.14,C_1,,)

T:dot(at Ct+(0,elen_))
   inductor(from T to Ct); dlabel(0.12,-0.1,,,L_1)

   Point_(-30); inductor(from Cr to Cr+vec_(elen_,0))
      dlabel(0,-0.07,,L_3,)
R:dot
L:dot( at (Cl-(Cos(30)*(elen_),0),R) )

   inductor(from L to Cl); dlabel(0,-0.12,,L_2,)

   right_; resistor(from L to R); rlabel(,R_2,)

   resistor(from T to R); dlabel(0,0.15,,R_3,) ; b_current(y,ljust)

   line from L to 0.2<L,T>
   source(to 0.5<L`',`'T>); dlabel(sourcerad_+0.07,0.1,-,,+)
      dlabel(0,sourcerad_+0.07,,u,)
   resistor(to 0.8<L`',`'T>); dlabel(0,0.15,,R_1,)
   line to T
.PE
```

The above source also illustrates that since **m4** macro arguments are separated by commas, any commas that are integral parts of the arguments must be protected either by parentheses as in `inductor(from Cr to Cr+vec_(elen_,0))` or by multiple single quotes, `` `',`' ``.

11 Logic gates

Fig. 10 shows a 3-input NOR gate, with the enclosing block and pre-defined internal locations `Out, In1`–`In3`. Gate macros have an optional argument, an integer from 0 to N where N can be up to approximately 5, defining locations `In1,` $\cdots$ `In`N. By default $N = 2$, except for macros

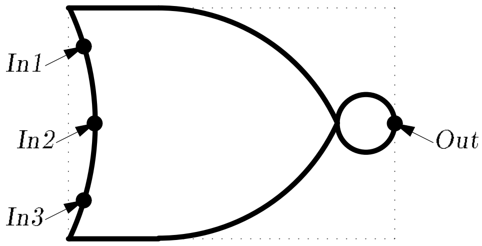

Figure 10: Illustrating `NOR_gate(3)`

`NOT_gate` and `BUFFER_gate` which have one input `In1`, unless they are given an argument, which is treated as the line specification of a two-terminal element. Fig. 11 shows these and the other basic

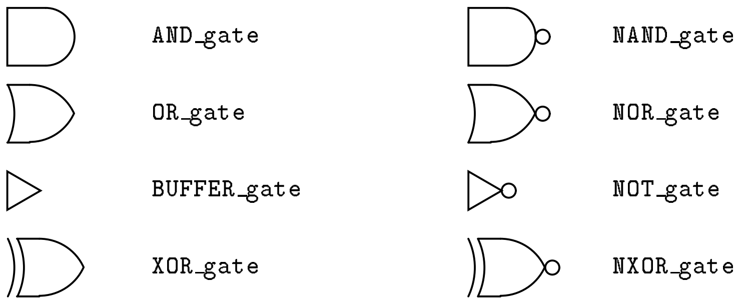

Figure 11: Basic logic gate library

logic gates included in library `liblog.m4`. These gates are typically not two-terminal elements and are normally drawn horizontally or vertically (although arbitrary directions may be set with e.g. `Point_(`*`degrees`*`)`). Each gate is contained in a block of typical height `6*L_unit` where `L_unit` is a macro intended to establish line separation for an imaginary grid on which the elements are superimposed.

The following source produces the SR-flip-flop shown in Fig. 12 to illustrate change of direction. Note that when a gate is rotated, its input locations retain their positions relative to the gate body.

```
.PS
log_init
S: NOR_gate
  left_
R: NOR_gate at S+(0,-L_unit*9)
  line right_ L_unit*3 from S.Out ; line to (Here,R.In2) then to R.In2
  line left_ L_unit*3 from R.Out ; line to (Here,S.In2) then to S.In2
  line left_ 4*L_unit from S.In1 ; "$S$sp_" rjust
  line right_ 4*L_unit from R.In1 ; "sp_$R$" ljust
.PE
```

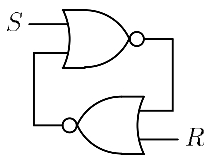

Figure 12: SR-flip-flop

A good strategy for drawing complex logic circuits might be summarized as follows:

- Establish the absolute locations of gates and other major components (e.g. chips) relative to a grid of mesh size commensurate with `L_unit`, which is an absolute length.

- Draw minor components or blocks relative to the major ones, using parametrized relative distances.
- Draw connecting lines relative to the components and previously-drawn lines.
- Write macros for repeated constructs.
- Tune the diagram by making absolute locations relative, and by tuning the parameters. Some useful macros for this are the following, which are in units of `L_unit`:

  `AND_ht, AND_wd`: the height and width of basic AND and OR gates

  `BUF_ht, BUF_wd`: the height and width of basic buffers

  `N_diam`: the diameter of NOT circles

A useful exercise might be to reproduce the example of Fig. 13.

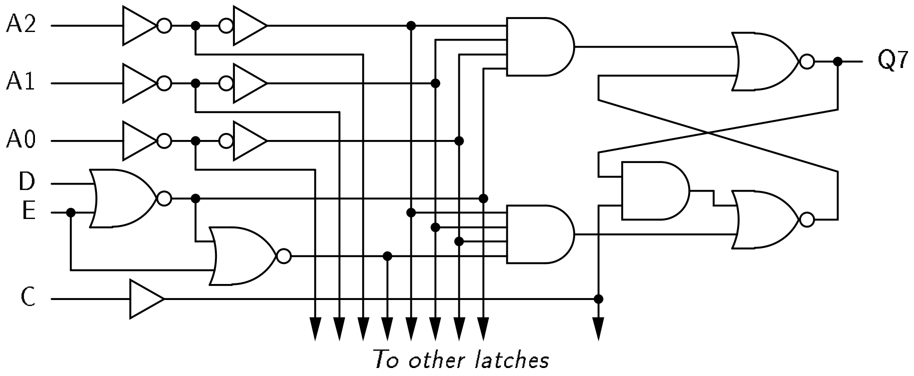

Figure 13: Multi-function latch circuit

12 Developer's notes

Because **gpic** or its equivalent were not then available, several years ago in the course of writing a book I took a few days off to write a **pic**-like interpreter **(dpic)** that produced latex picture objects. More recently the interpreter has been upgraded to generate **mfpic** or **PSTricks** commands, the latter my preference because of the quality and flexibility of resulting graphics, including facilities for colour and rotations. Very recently, Fig 3.1 output has been added. I preferred the more-powerful **m4** macro processor to **pic** macros, and therefore **m4**, which comes with many Unix systems and a free version of which is available, is required here. If starting over today would I not just start with the **gpic** source and change its back end? Good question. Maybe. Another question might be, why not use **PSTricks** (or **mfpic**) alone? Much can be done that way using macros, but the **pic** language is extremely simple to learn, especially for backslashophobics like me. The language has stood the test of time, it has built-in looping and block-structure constructs like

```
drawing commands ...
[drawing commands...
   A:[...
      B:box
     ]
] with .A.B.bottom at 2nd last ellipse.top
```

that combine power with simplicity, and it allows the simple computation of geometries using elementary functions.

Gpic and **dpic** pass unaltered any line beginning with a \ character, allowing TeX or PSTricks macros to be invoked, for example to set parameters such as colour or fill values.

Using the **mfpic** output of **dpic** it is possible to produce Metafont alphabets of circuit elements or their sub-components, thereby essentially removing dependence on device drivers, but with the complication of treating every graphic sub-component as a TEX box.

The **xfig** output of **dpic** allows elements to be defined and fine-tuned, and then using the interactive graphics of **xfig,** quickly assembled into a circuit. However, the elements should now have all boundary points and points of connection on the grid points, or at least the grid lines, of the drawing canvas. Further refinement of the elements might still be required.

Because the set of common circuit components and potential macros is huge, an industrial-strength system developed from the one described here would probably require a combination of all of the techniques described, including the ability to dynamically define and build character sets from macros.

13 Bugs

Initialization: If the first element macro evaluated is non-two-terminal or is within a Pic block, then later macros evaluated outside the block may produce the error message

```
there is no variable ‘rp_ang’
```

because `rp_ang` is not defined in the outer-most scope of the diagram. To cure this problem, put the line

```
cct_init
```

immediately after the .PS line, or prior to the first block. It is entirely permissible to modify `cct_init` to include commonly-used diagram initializations, such as the `thicklines_` statement, and to invoke cct_init at the beginning of every diagram. For completeness, macros `gen_init, log_init, darrow_init` are also provided for cases where the circuit library is not included.

Commas: Remember that macro arguments are separated by commas, so commas which are part of an argument must be protected by parentheses or quotes. Thus,

```
shadebox(box with .n at w,h)
```

produces an error, whereas

```
shadebox(box with .n at w‘,’h)
```

and

```
shadebox(box with .n at (w,h))
```

do not.

Names: Using the name of a macro as part of a comment or string is a simple and common error. Thus,

```
arrow right "$\dot x$" above
```

produces an error message because `dot` is a macro name. Macro expansion can be avoided by adding quotes, as follows:

```
arrow right ‘"$\dot x$"’ above
```

To avoid name conflicts, all library macros intended only for internal use have names that begin with `m4`.

A good rule is to process diagrams in separate files, and to use the TeX `\input` command to include the result. If extensive use of strings that conflict with macro names is required, then another solution is to replace the strings by macros which are expanded by LATEX , for example the diagram

```
.PS
   box "\stringA"
.PE
```

with the LaTeX macro

```
  \newcommand{\stringA}{%
   Circuit containing planar inductor and capacitor}
```

Current direction: Some macros, particularly those for labels, do unexpected things if care is not taken to pre-set the current direction using macros `right_`, `left_`, `up_`, `down_`, or `rpoint_(·)`. Thus for two-terminal macros it is good practice to write, e.g.

`resistor(up_ from A to B); rlabel(,R_1)`

rather than

`resistor(from A to B); rlabel(,R_1)`,

which produce different results if the last-defined drawing direction is not up. It might be possible to change the label macros to avoid this problem without sacrificing ease of use.

14 List of macros

The following table lists the macros in libraries darrow.m4, libcct.m4, liblog.m4, libgen.m4, and files gpic.m4, mfpic.m4, and pstricks.m4.

Internal macros defined within the libraries begin with the characters m4, and are not listed here.

The libraries use the following **pic** variables:

```
m4dll
m4dlw
m4i
m4j
m4t1
m4t2
m4tmp
rp_ang
rp_ht
rp_len
rp_wid.
```

The library in which each macro is found is shown, and a brief description.

`AND_gate(`*n*`)`	log	basic 'and' gate, 2 or *n* inputs
`AND_ht`	log	height of basic 'and' and 'or' gates
`AND_wd`	log	width of basic 'and' and 'or' gates
`BUFFER_gate(`*linespec*`)`	log	basic buffer, 1 input or as a 2-terminal element
`BUF_ht`	log	basic buffer gate height
`BUF_wd`	log	basic buffer gate width
`Cos(`*integer*`)`	gen	cosine function, *integer* degrees
`E_`	gen	the constant e
`G_hht_`	log	gate half-height
`HOMELIB_`	all	directory containing libraries
`L_unit`	log	logic-element grid size
`NAND_gate(`*n*`)`	log	'nand' gate, 2 or *n* inputs
`NOR_gate(`*n*`)`	log	'nor' gate, 2 or *n* inputs
`NOT_gate(`*linespec*`)`	log	'not' gate, 1 input or as a 2-terminal element
`NXOR_gate(`*n*`)`	log	'nxor' gate, 2 or *n* inputs
`N_diam`	log	diameter of 'not' circles

OR_gate(*n*)	log	'or' gate, 2 or *n* inputs
Point_(*integer*)	gen	sets direction cosines in degrees
Rect_(*radius*,*angle*)	gen	(deg) polar-to-rectangular conversion
Sin(*integer*)	gen	sine function, *integer* degrees
XOR_gate(*n*)	log	'xor' gate, 2 or *n* inputs
above_	gen	string position above relative to current direction
abs_(*number*)	gen	absolute value function
amp(*linespec*)	cct	amplifier
battery(*linespec*,n)	cct	n-cell battery, default 1 cell
below_	gen	string position relative to current direction
bi_tr(*linespec*,L|R,P,E)	cct	left or right, N or P-type bipolar transistor, without or with envelope
b_current(*label*,*pos*,*dir*)	cct	draw and label branch-current arrow
capacitor(*linespec*,C)	cct	capacitor, straght or curved-plate
clabel(*label*,*label*,*label*)	cct	centre triple label
consource(*linespec*,V|I)	cct	blank or voltage or current-controlled source
cross(at *location*)	gen	plots a small cross
crosswd_	gen	cross dimension
csdim_	cct	controlled-source width
d_fet(*linespec*,L|R,P,E)	cct	left or right, N or P depletion MOSFET, without or with envelope
darrow(*linespec*,t,t,*width*)	darrow	double arrow
delay(*linespec*)	cct	delay element
delay_rad_	cct	delay radius
dend(at *location*)	darrow	close (or start) double line
diff_(a,*b*)	gen	difference function
dimen_	cct	size parameter for circuit elements
dimension_(*linespec*,*offset*,*label*,*label wid*,*tic offset*)	gen	macro for dimensioning diagrams
diode(*linespec* ,Z|BD)	cct	diode, normal or zener or bi-directional
dlabel(*long*,*lat*,*label*,*label*,*label*)	cct	general triple label
dleft	darrow	double line left turn
dline(*linespec*,t,t,*width*)	darrow	double line
dlinewid	darrow	width of double lines
dn_	gen	sets down relative to current-direction
dot(at *location*)	gen	draw a dot
dotrad_	gen	dot radius
down_	gen	sets current direction to down
dright	darrow	double arrow right turn
dtee('*direction*')	darrow	double arrow tee junction
dtor_	gen	degrees to radians conversion constant
e_	gen	.e relative to current direction
e_fet(*linespec*,L|R,P,E)	cct	left or right, N or P enhancement MOSFET, without or with envelope
ebox(*linespec*,*length*,*ht*)	cct	two-terminal box element with adjustable dimensions
eleminit_(*linespec*)	cct	internal line initialization
elen_	cct	default element length
fill_(*number*)	gen	fill macro, 0=black, 1=white
gamp(*linespec*)	cct	general amplifier
gap(*linespec*)	cct	gap with dots
glabel_	cct	internal general labeller
gpic_	gpic	defined to signify gpic is being used
grid_(*x*,*y*)	log	absolute grid location
ground(at *location*,T)	cct	ground, without stem for nonblank 2nd arg
hop(L|R,at *location*)	cct	conductor crossing another to left or right

hoprad_	cct	hop radius
ht_	gen	height relative to current direction
inductor(*linespec*,W,*n*,M)	cct	inductor, narrow or wide, 4 or *n* arcs, without or with magnetic core
integrator(*linespec*)	cct	integrating amplifier
j_fet(*linespec*,L|R,P,E)	cct	left or right, N or P JFET, without or with envelope
left_	gen	left with respect to current direction
linethick_(*number*)	gen	set line thickness in points
ljust_	gen	ljust with respect to current direction
llabel(*label*,*label*,*label*)	cct	triple lable on left side of the element
loc_(*x*, *y*)	gen	location adjusted for current direction
lt_	gen	left with respect to current direction
manhattan	gen	sets direction cosines for left, right, up, down
mfpic_	mfpic	defined to signify mfpic is being used
n_	gen	.n with respect to current direction
ne_	gen	.ne with respect to current direction
neg_	gen	unary negation
nw_	gen	.nw with respect to current direction
opamp(*linespec*,*label*,*label*)	cct	operational amplifier with −, + or other internal labels
point_(*angle*)	gen	(radians) set direction cosines
polar_(*x*,*y*)	gen	rectangular-to polar conversion
prod_(a,*b*)	gen	binary multiplication
project(*x*,(*y*,(*z*)	gen	3D to 2D projection
psset_(*PSTricks settings*)	gen	set PSTricks parameters
pstricks_	pstricks	defined to signify PSTricks is being used
pt_	gen	big point size
rect_(*radius*,*angle*)	gen	(radians) polar-rectangular conversion
resistor(*linespec*,n)	cct	resistor, n peaks, default 3
right_	gen	set current direction right
rjust_	gen	right justify with respect to current direction
rlabel(*label*,*label*,*label*)	cct	triple lable on right side of the element
rpoint_(*angle*)	gen	(radians) set direction cosines
rpos_(*position*)	gen	Here + *position*
rt_	gen	right with respect to current direction
rtod_	gen	constant, degrees/radian
rvec_(*x*,*y*)	gen	location relative to current direction
s_	gen	.s with respect to current direction
se_	gen	.se with respect to current direction
setview(*azimuth*,*(radians) elevation*)	gen	set projection viewpoint
shadebox(*box specification*)	gen	box with edge shading
sign_(*number*)	gen	sign function
source(*linespec*,V|I|AC|*string*)	cct	source, blank or voltage or current or AC or labelled
sourcerad_	cct	source radius
sp_	gen	evaluates to medium space for gpic strings
sum_(a,*b*)	gen	binary sum
svec_(*x*,*y*)	log	scaled and rotated grid coordinate vector
sw_	gen	.sw with respect to current direction
switch(*linespec*,L|R,C|O)	cct	SPST switch left or right, blank or closing or opening arrow
thicklines_(*number*)	gen	set line thickness in points
thinlines_(*number*)	gen	set line thickness in points
transformer(*linespec*,L|R,n)	cct	2-winding transformer, left or right, n arcs
twopi_	gen	2π

`up_`	gen	set current direction up
`up__`	gen	up with respect to current direction
`vec_(`*x*`,`*y*`)`	gen	position rotated with respect to current direction
`vrot_(`*x*`,`*y*`,`*xcosine*`,`*ycosine*`)`	gen	rotation operator
`vscal_(`*number*`,`*x*`,`*y*`)`	gen	vector scale operator
`w_`	gen	.w with respect to current direction
`wid_`	gen	width with respect to current direction
`xtal(`*linespec*`)`	cct	quartz crystal

References

[1] Graphic symbols for electrical and electronic diagrams, 1975. IEEE Std 315-1975.

[2] J. Bentley. *More Programming Pearls.* Addison-Wesley, New York, 1988.

[3] The Free Software Foundation. Gpic man page, 1992.

[4] B. W. Kernighan and D. M. Richie. The M4 macro processor. Technical report, Bell Laboratories, 1977.

[5] B. W. Kernighan and D. M. Richie. PIC—A graphics language for typesetting, user manual. Technical Report 116, AT&T Bell Laboratories, 1991.

[6] T. Leathrum. Pictures in TEX with metafont, 1992.

[7] T. Rokicki. DVIPS: A TEX driver. Technical report, Stanford, 1994.

[8] T. Van Zandt. PSTricks user's guide, 1993.