Circuit_macros

An application of little languages

Keywords
Electric circuit diagrams, line drawings, LaTeX

Abstract
The evolution of the Circuit_macros package is described, with some of the conventions for drawing circuit elements and some of the lessons learned.

Introduction

Adding diagrams and other artwork to TEX and LATEX documents is a source of guaranteed discussion among users of these tools. There is a large selection of programs capable of creating artwork in .eps form and a somewhat smaller group that invoke the TEX engine as part of the drawing process. The circuit_macros tools are in the second group. This article briefly describes the macros provided in the package, along with some decisions taken during their evolution and some lessons learned.

In 1989 I needed to produce a few simple diagrams, including Figure 1, for a research monograph [1] that I was producing on a basic LATEX installation. From the perspective of the year 2004 it is hard to remember that there were few graphic tools readily available at that time. The picture environment mentioned in the LATEX manual [5] seemed adequate, but the coordinate calculations were tedious, and I automated them by writing an interpreter. I called it dpic, since it implemented a subset of the pic language [4]. The original pic language was developed at a time when a typist would be given hand-written notes with a sketch or two and told to put them "into the computer", so simplicity was its primary feature, together with moderate drawing power. The resulting language is very easy to learn and read.

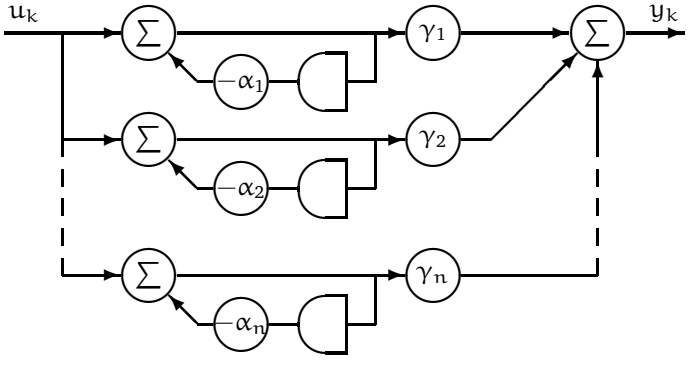


Figure 1. An operational diagram produced by careful use of LATEX picture objects and the epic style.

I also needed to draw basic electric and electronic circuits, but found little support for including them in LaTeX documents. It was a simple step to supplement the basic line drawing of the pic language with circuit-element macros defined in the widely available m4 macro language. However, the LaTeX picture objects produce only limited line slopes and lengths, so alternative processing is required. One solution is the GNU pic processor gpic [6] that generates tpic `\special` commands, but it was also easy to modify dpic to generate output in several formats, of which the PSTricks [7] package seemed to offer the most power and flexibility at the time, at least for producing Postscript files. The current Circuit_macros distribution and the dpic interpreter are the result of those efforts, drawings I have had to produce since, and suggestions received. The macro libraries, example files, and user's manual [2] total approximately 12 000 source lines. The overall philosophy has been to combine the power of proven components implementing "little languages", as encouraged in the Unix computing environment. Thus the workflow has a Unix flavour, but is readily performed on other operating systems, such as a Windows machine with the Cygwin tools installed. A "make" facility, for example, can automate the complete production of a book containing scores of diagrams.

An overview will be given of the macros, together with some of the processing possibilities that the dpic interpreter offers. Element design is emphasized, rather than the details of the pic [6] or m4 [3] languages. For more information on the macros, the user's manual [2], available in any CTAN repository in `graphics/Circuit_macros`, can be consulted.

In the following, keep in mind the pic drawing elements: linear objects (line, arrow, move, spline, arc), and planar objects (box, ellipse, circle). Planar blocks containing arbitrary objects can be defined. The start, centre, and end of linear objects and the compass corners of planar objects can be referenced. Strings to be typeset in a later word-processor step are allowed. Variables beginning with lowercase letters can be assigned values and used in computations. Any drawn element can be given a name beginning with an upper-case letter. The scope of a variable is the block that contains it, but positions can be referenced from outside a block. Pic also has a selection of mathematical functions, as well as looping and other facilities.

A definition in the m4 macro language is of the form `define(‘`*name*`’,` *replacement text*`)`, and the macro is invoked with comma-separated arguments as *name*`(`*arg 1*`,` *arg 2*`, ...)`. Strings are protected from expansion by enclosing them in `‘, ’` quotes, and recursive macro expansion is allowed.

Workflow alternatives

The essentials of the process to be described are shown in Figure 2. The choice of output formats of dpic is provided both for compatibility and necessity.

Dpic without options or with the `-t` option is for producing simple, portable diagrams. The Mfpic output is for compatibility, for creating fonts containing portable diagrams, or as an intermediate step in producing pdf files. The MetaPost output is provided for compatibility and because this is also a good route to producing pdf files on many installations.

The xfig output of dpic allows the definition of detailed circuit elements in m4 and pic form, and their interactive placement using xfig. Xfig can output diagrams in pic format, so an iterative design process is sometimes appropriate.

Some TeX and Metafont programs have fixed-length storage arrays, which can be over-filled by diagrams containing many drawn objects. The .eps format with psfrag strings is an alternative but also occasionally exceeds program capacity. The raw Postscript output of dpic removes any requirement for passing the drawing through LaTeX, but the text labels are not then typeset. Very large or complex diagrams can be produced by creating *two* versions, one containing only graphics converted

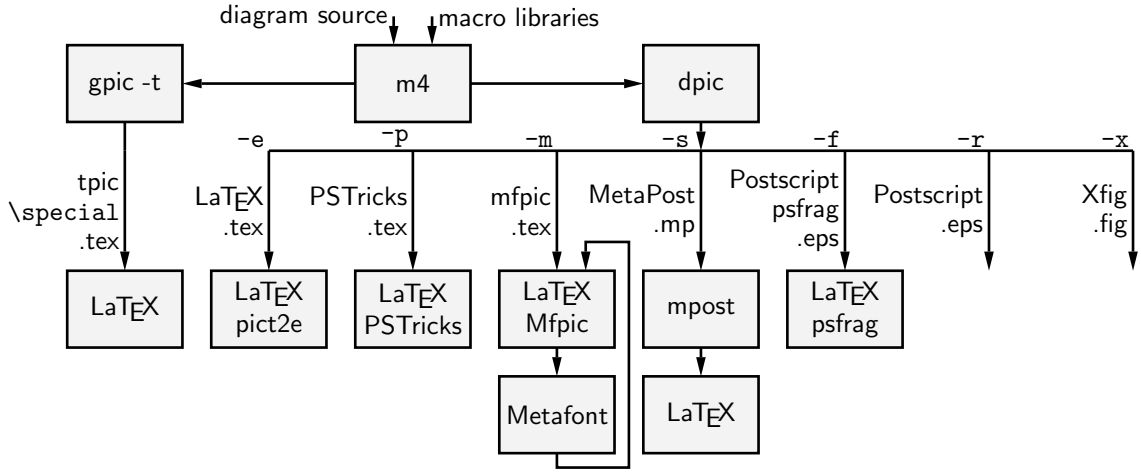


Figure 2. The Circuit_macros workflow. The dpic options and filetypes produced are shown. In some cases, TEX or PDFLatex can be substituted for LaTEX. Strings are not typeset in the raw Postscript or xfig output of dpic.

to .eps form by dpic, and the other containing only labels at appropriate places, overlaying the first diagram in the final document.

For producing Postscript, the PSTricks option has proven both flexible and reliable, and can be assumed by default in the following descriptions. It can be supplemented by `ps4pdf` or sometimes the PDFtricks package.

What is a circuit?

Basic circuits contain two-terminal elements, but multiterminal elements are allowed. In fact, a circuit diagram can be a rather general line diagram together with typeset text. Color, elaborate fills, and visual transformations associated with pictorial graphics are comparatively rare, however. Therefore, almost any line-drawing tool can produce circuits. Regardless of language, effort is required to design high-quality standard elements that are easy to use and modify.

No library of drawing objects can hope to satisfy the needs and taste of everyone, and the potential number of circuit elements is huge. Therefore, the Circuit_macros manual encourages the user to modify and add to the published macros, and some attention has been paid to ease of modification in their design. However, there is always a compromise between macro generality and simplicity.

Two-terminal elements

In the Circuit_macros collection, two-terminal elements are drawn according to a common convention. The first argument of each macro defines an invisible line along which the element is drawn; thus, for example, `inductor(up_ 0.5 from A)` draws an inductor 0.5 units up from predefined position A. All macro arguments are optional, and when the first argument (called a *linespec*) is omitted, a line of default length in the current drawing direction is assumed.

An invisible line that can be named is first drawn as specified by the *linespec*. Then the visible components are added, an invisible block is placed over the element body to facilitate later labeling, and another invisible line is finally drawn to allow post-reference to the element. For example, the command `L1: inductor(,W)` produces the inductor shown in Figure 3, the second argument specifying a “wide” body shape with a default number of loops. The element (the initially drawn invisible line, in fact) has been named L1, so the positions L1.start, L1.center, and L1.end can later be referenced.

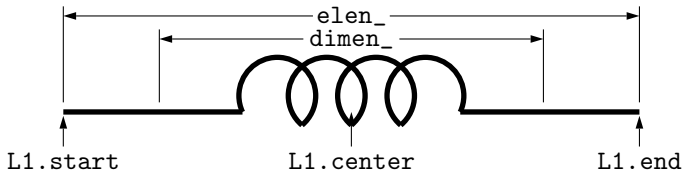


Figure 3. Inductor produced by `L1:inductor(,W)`.

Figure 3 shows the length of the macro `dimen_`, which evaluates to the dpic parameter `linewid`. Element bodies are drawn in units of `dimen_`, so changing its definition or assigning a new value to `linewid` changes the size of elements. The macro `elen_` is the default element length and is initially set to `dimen_*3/2`.

Depending on the element, the arguments may be used to specify element type and other variations. Figure 4 shows the diode variants, for example. In addition to the radiation arrows shown, a variety of arrows and marks is available to signify element variability.

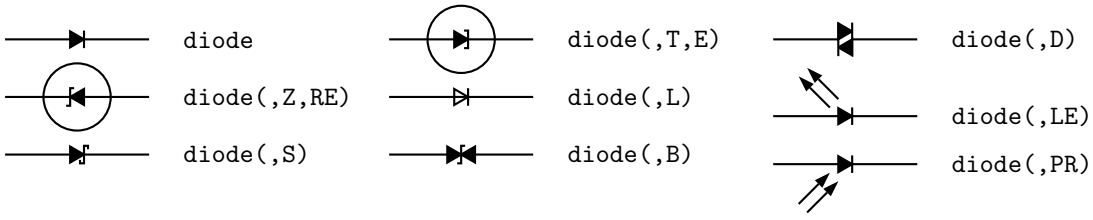


Figure 4. Variants of the macro `diode(`*linespec*`,B|D|L|LE[R]|P[R]|S|T|Z,[R][E])`.

Multiterminal elements

By convention, each multiterminal elements is enclosed in a block. Connection nodes and important sub-elements are named. If a *linespec* is one of the arguments, it defines the reference direction and length of the element but not its position, since the enclosing block is positioned by default or by specifying the location of one of its elements. Thus, for example, `bi_tr with .B at Here` draws a bipolar transistor in the current drawing direction, placed with internal location B (the base connection) at the current drawing location `Here`, which is always defined. Multiterminal elements are often asymmmetric with respect to the drawing direction and typically contain an argument to orient them to the right or left. Thus, Figure 5 shows commands to orient a transistor to left and right, and the resulting diagram.

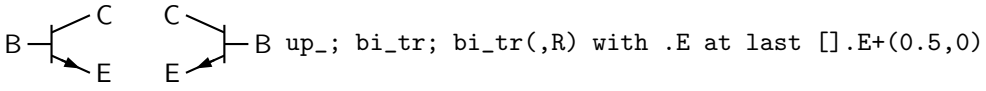
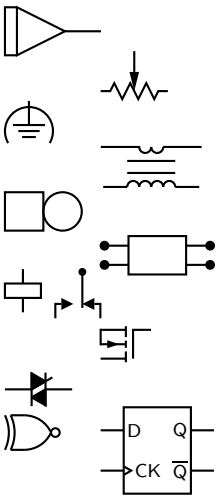


Figure 5. The macro `bi_tr(`*linespec*`,L|R,P,E)` contains internal locations E, B, and C. The figure shows both left and right orientation of the transistor base with respect to the upward drawing direction.

The libraries

In addition to two-terminal elements, the circuit libraries currently contain definitions for the following, with representative samples shown:



- □ operational amplifiers, delays, and integrators
- □ potentiometers,
- □ grounding and protection symbols,
- □ transformers,
- □ audio components: speaker, bell, microphone, buzzer, earphone,
- □ general n-ports,
- □ contacts and relays,
- □ bipolar and MOS devices,
- □ scr devices,
- □ functional and rectangular logic gates, and
- □ flip-flops and multiplexors.

The distribution contains examples of flowcharts, binary trees, signal-flow graphs, parallel-line arrows, and integrated circuits. In total, there are approximately 600 macros, a minority of which are meant to be called internally by other macros.

Changing direction

Circuit elements all have a reference direction, and most have an orientation (left or right) with respect to this direction. Most elements can be drawn up, down, left, right or, if required, at any angle. Labels typically should not rotate with the element.

Although pic provides several ways of placing an element, it does not provide arbitrary rotation, which must be performed instead at the macro expansion stage. A transformation matrix similar to the Postscript coordinate transformation matrix is defined each time an element *linespec* or its default is executed, and the element is rotated accordingly. The reference direction can also be set explicitly.

By convention, all macros are designed with respect to a reference direction pointing right (or 0 degrees). Coordinates are specified as `vec_(`*x*`,`*y*`)` and relative coordinates as `rvec_(`*x*`,`*y*`)`. These two macros expand to the required rotated and translated coordinates.

Customizing elements

Recent versions of the libraries have attempted to simplify the customization of defined elements. The diode macro shown in Figure 4 is an example. This macro could simply test its arguments and invoke one of many special diode macros, but this solution was proving cumbersome and hard to maintain. Instead, the diode macro now draws the initial invisible line, sets some internal dimensions, and invokes the internal macro `m4gen_d(`*chars*`)` that draws the details. Its argument is a "DNA-like" character string. This string is tested for a given substring, and if present, the substring is removed, the corresponding element part is drawn, and the process is repeated for other substrings. Thus for example, the argument of `m4gen_d(LCFR)` contains L to draw the left stem of the diode, C to draw the cathode crossbar, F for the filled arrowhead, and R for the right stem. Custom elements are then easy to define by omitting argument substrings or modifying the macros

to test for new substrings and draw the corresponding subcomponents.

Time will tell whether this DNA-like solution is flexible and robust. It provides easy customization but requires the user to understand it.

Interaction with LaTeX and post-processors

Often it is necessary to know the dimensions of typeset text in a figure, but these dimensions are unavailable until after LaTeX is run on the document containing the diagram. This classic forward-referencing problem can be handled just as LaTeX handles other references, by writing definitions to a file and processing twice.

The distribution includes a small style file, `boxdims.sty`, defining a LaTeX macro `\boxdims` that inserts the dimensions of its second argument into the definition of an m4 macro named by the first argument. Such definitions are written to the file *jobname*`.dim`, where *jobname* is the main LaTeX file, or to an explicitly named file. On the second pass, m4 reads in the `.dim` file and, *voilà*, the dimensions of the typeset text are known.

The pic language allows arbitrary strings to be inserted into its output. For example, these strings can command PSTricks, MetaPost, or Postscript to displace, fill, clip, or rotate objects, including their typeset labels.

Comments

The Circuit_macros library has evolved over a decade and a half. Circuits can be (and are) produced in any of several excellent drawing languages now available, although comments I have received testify to the ease of learning pic. Drawing tools are only part of the issue, however; good design requires thought and work no matter what mechanism places the lines on the document.

References

[1] J. D. Aplevich. *Implicit Linear Systems*. Springer Verlag, Heidelberg, 1991. Lecture Notes in Control and Information Sciences, Vol. 152.

[2] J. D. Aplevich. M4 macros for electric circuit diagrams in LaTeX documents, Circuit_macros user's guide, 2004.

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

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

[5] L. Lamport. *LaTeX, A Document Preparation System*. Addison-Wesley, Reading, Mass., 1986.

[6] E. S. Raymond. Making pictures with GNU PIC, 1995. In GNU groff source distribution.

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

Dwight Aplevich
University of Waterloo Waterloo Canada