Notes on Drawing Logic Diagrams with LaTeX

Adrian P. Robson

29 January 2020

These are notes on some methods for drawing logic gate diagrams in LaTeX with its Ti*k*Z package. The `circuits.logic.us` and `positioning` libraries are used.

Contents

1 Drawing Connectors

Connectors should be composed of only vertical or horizontal lines for good style. There are two possibilities for a point to point connector: straight or stepped.

1.1 Stepped Connectors

To draw a stepped connector, a coordinate with a x-value that gives the location of the vertical part of the connection is used. The coordinate's

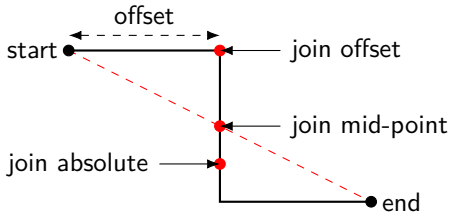


Figure 1: Stepped Simple Connector

y-value does not have to be precise, but it must be between the y-values of the start and end points.

This coordinate is used to draw the connection as follows, where it is called `join`:

```
1 \draw (start) -| (join) (join) |- (end);
```

This draws the connector as two right angled lines meeting at the `join` coordinate.

The position of the `join` coordinate can be defined using absolute, offset or mid-point methods, as illustrated in figure 1, which shows the three possible `join` coordinates.

The absolute method is implemented with:

```
1 \coordinate (join) at (1,-0.5);
```

This defines a coordinate called `join` at an absolute position. Its x-value gives the horizontal location of the connector's vertical part. *Its y-value must be between the vertical part's top and bottom points.*

The offset method is a little more complicated:

```
1 \def\relx{1}
2 \path (start) ++(right:\relx) coordinate (join);
```

Line 1 defines a horizontal offset value. It must be less than the connector's available horizontal space.

Line 2 creates a coordinate that is shifted right from the `start` coordinate. Its x-value is the horizontal position of the connection's vertical part. Its y-value is always within its acceptable range.

The mid-point method is simple:

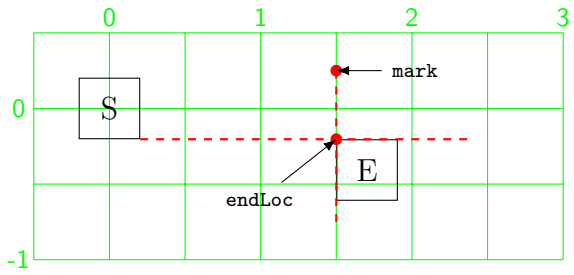


Figure 2: Straight Simple Connectors

```
1 \path (start) -- coordinate(join) (end);
```

This creates a coordinate horizontally half way between the connections start and end points. Its x-value and y-value are always within acceptable limits.

1.2 Straight Connections

For straight connections the start and end connection points have to be aligned. If a node's connection point is off its centre, then simple relative or absolute node alignment will not work. Unfortunately, this is is the case for most logic gate inputs.

Figure 2 illustrates a method for aligning connector end points. It uses square nodes instead of gates for simplicity. It shows the horizontal alignment of a node's south east corner and another node's north west corner.

In the following example, the actual connection is not drawn. An absolute method is used, but it could be easily adapted to use a relative offset to place the end node. The relevant code is:

```
1 \node at (0,0) [draw,minimum size=8mm] (start) {S};
2 \coordinate (mark) at (1.5,0.25);
3 \coordinate (endLoc) at (start.south east -| mark);
4 \node [draw,minimum size=8mm,anchor=north west]
5             (end) at (endLoc) {E};
```

Line 1 makes a rectangle node called `start` at an absolute position.

Lines 2 defines a coordinate called `mark` at an absolute position. Its x-value is the horizontal position of the end connection point. Its y-value is not important, but it should be given a reasonable value to help with possible debugging.

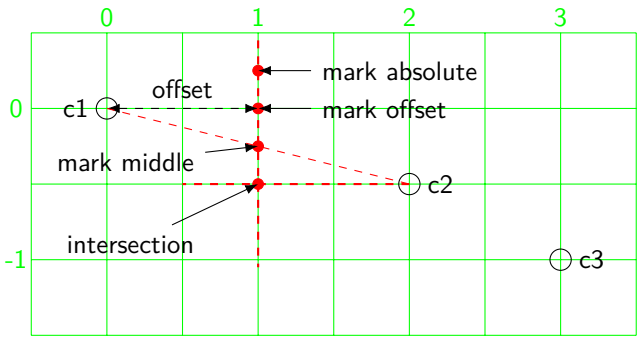


Figure 3: Splitting Connectors – Methods

Line 3 calculates the required position of `end` node's `end.north west` anchor. It uses TikZ's intersection operator (see appendix A), which here finds the intersection of a line the goes vertically through `mark` and the horizontal line that goes through the anchor `start.south east`, and calls it `endLoc`.

Line 4–5 makes a rectangular node called `end`. It is positioned so that its `north west` anchor is in the same place as `endLoc` calculated above.

Thus, a line connecting `start.south east` and `end.north west` will be horizontal; and can be drawn with the following:

```
1 \draw (start.south east) -- (end.north west);
```

1.3 Splitting Connectors

Connecting the output of a gate to the inputs of more than one other gate is common. Here are absolute, offset and mid-point methods for doing this. Figure 3 illustrates how a split point can be calculated using these methods. They are, in part, the same as those used for stepped connectors. The following discussion assumes that all of the gates have been previously drawn, and a split connector is needed between point `c1` and both `c2` and `c3`.

1. First a coordinate called `mark` that defines the position of the connector's vertical part is defined. The following methods are the same as those used in §?? for drawing stepped connectors.

   The absolute method is very simple:

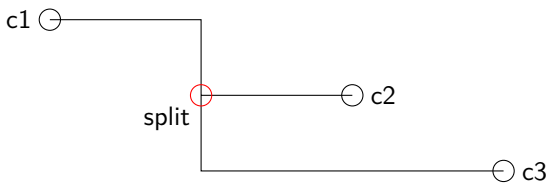


Figure 4: Splitting Connectors – Lines

```
\coordinate (mark) at (1,0.25);
```

This defines `mark` at an absolute position. Its x-value gives the horizontal location of the connector's vertical part. *Its y-value must be between the vertical part's top and bottom points.*

The offset method is implemented with:

```
\def\relx{1}
\path (c1) ++(right:\relx) coordinate (mark);
```

Here the `\relx` value gives the horizontal offset of `mark` from point `c1`.

The mid-point method puts the split point half way between the connected nodes. This is done with

```
\path (c1) -- coordinate(mark)  (c2);
```

This creates `mark` half way between nodes `c1` and `c2`.

2. Next the position of the split point is calculated:

```
\coordinate  (split) at (mark |- c2);
```

This uses TikZ's intersection operator (see appendix A), which here finds the intersection of a line the goes vertically through the `mark` coordinate and the horizontal line that goes through the point `c2`.

3. Once the split point is established, the connector can be drawn though this coordinate:

```
\draw (c1) -| (split) (split) -- (c2)
              (split) |- (c3);
```

This draws three lines connecting the nodes to the split point. Figure 4 shows the diagram with lines added.

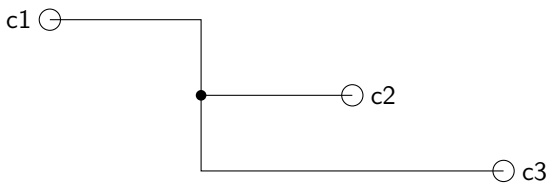


Figure 5: Splitting Connectors – Split Dot

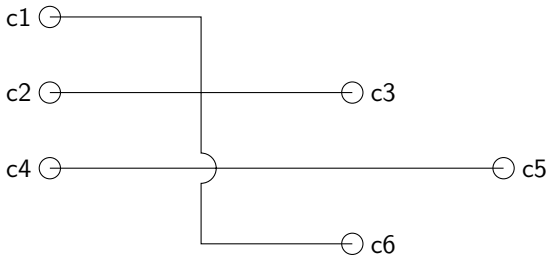


Figure 6: Crossing Points

1.4 Dots and Jumps

A dot can be put at the split point as shown in figure 5. First the following is put in the document's preamble:

```
\newcommand{\splitmark}[2]{(#1)
           node[circle, fill, inner sep= 0pt, outer sep= 0pt,
                              minimum size= 2*#2]{}%
}
```

This macro takes two parameters: The first is a coordinate that gives the location of the split point, and the second is the radius of the dot. The `splitmark` macro is used in the path that draws the connection where `split` is the split point, like this:

```
   \draw (c1) -| \splitmark{split}{2pt} -- (c2)
                 (split) |- (c3);
```

Jumps can be added to lines at crossing points as shown in figure 6, where there is also a non-jumping crossing for comparison. In this example, the jump is to the right on the diagram's down going vertical connector, but macros are given for all cardinal directions. Put the following in the preamble:

```
\newcommand{\jumpD}[2]{([shift={(0mm,#2)}]#1)
           arc[start angle= 90, end angle= -90, radius= #2]%
}
\newcommand{\jumpU}[2]{([shift={(0mm,-#2)}]#1)
           arc[start angle= -90, end angle= 90, radius= #2]%
}
\newcommand{\jumpR}[2]{([shift={(-#2,0)}]#1)
           arc[start angle= 180, end angle= 0, radius= #2]%
}
\newcommand{\jumpL}[2]{([shift={(#2,0)}]#1)
           arc[start angle= 0, end angle= 180, radius= #2]%
}
```

These macros take two parameters: The first is a coordinate that gives the location of the jump point, and the second is the radius of the jump arc. They use coordinate shift notation to align the jump arc with the crossing point. With the macros in the preamble, the jump shown in figure 6 can be drawn with:

```
1 \coordinate (mark) at (1,0.25);
2 \coordinate (crosspoint) at (mark |- c5);
3 \draw (c2) -- (c3) (c4) -- (c5)
4                    (c1) -| \jumpD{crosspoint}{1mm} |- (c6);
```

Line 1 defines a coordinate that give the horizontal position of the jump point. Its y-value is not used. It is called `mark`.

Line 2 calculates the intersection point of a vertical line through `mark` and a horizontal line through `c5`. It is called `crosspoint`.

Line 3 draws two horizontal connectors.

Line 4 draws a stepped connector between `c1` and `c6`. The `jumpD` macro draws the jump arc around `crosspoint`.

2 Drawing Diagrams

2.1 Drawing Gates

To draw logic gate diagrams, the following should be put in the document preamble:

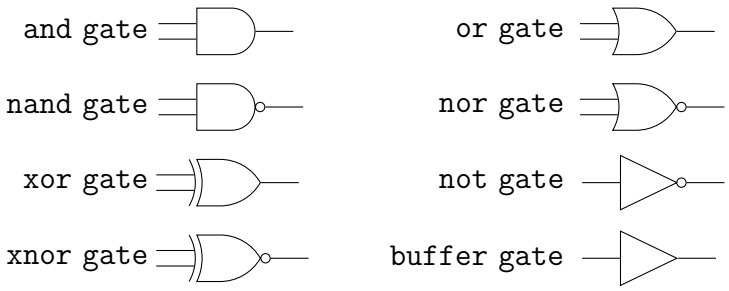


Figure 7: Gate Node Shapes

```
\usepackage{tikz}
\usetikzlibrary{circuits.logic.US}
\usetikzlibrary{positioning}
```

The `circuit logic US` option, which gives logic gates a 'distinctive' style[1], should be used for the `tikzpicture` environment:

```
\begin{tikzpicture}[circuit logic US]
...
\end{tikzpicture}
```

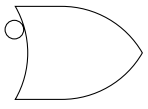
Gates are drawn as nodes. The available shapes are shown in figure **??**. So to draw an AND gate at an absolute position the following could be used:

```
\node [and gate] (and1) at (1,3) {};
```

This produces a gate with two normal inputs. If more inputs or negated inputs are wanted, use the `inputs` option. The following, which are equivalent, draw an OR gate with three inputs, the top one of which negated:

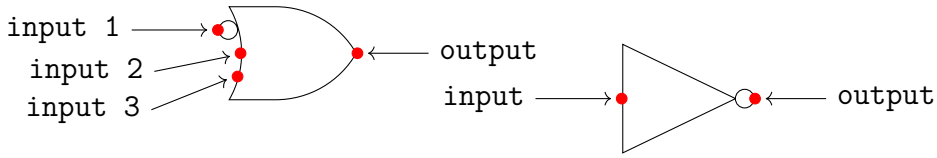
```
\node [or gate,inputs={inverted,normal,normal}] (or1) {};
\node [or gate,inputs=inn] (or1) {};
```

The gate looks like this:



[1]IEEE Std 91/91a

Gate nodes have anchors for their output and inputs. The output anchor is named `output`, and for multi-input gates, the inputs are named `input` n, where n is an integer. The top input anchor is `input 1`, the next down is `input 2` and so on. So the bottom input of a three input gate called `lastgate` would be called `lastgate.input 3`. Single input gates, like `not gate` and `buffer gate`, can have only one input anchor, and it is called `input`.



2.2 Input Spacing

The standard key `logic gate input sep` unfortunately changes the size of the gate symbol. An alternative is to use a dummy input to move the inputs apart without changing the gate's size:

Default

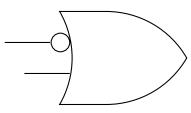
With dummy input

```
\node [and gate,inputs=nnn] (and2) at (0,0) {};
\draw (and2.input 1) -- ++(left:4mm);
% and2.input 3 is dummy
\draw (and2.output)  -- ++(right:4mm);
```

2.3 Aligning Input and Output End Points

Neatly positioning the ends of a logic gate diagram's input and output lines is difficult. Absolute coordinates do not work because they cannot be accurately vertically aligned with gate anchors. Relative positioning does not cope with negated inputs, as the following demonstrates:

```
\node [or gate, inputs={inverted,normal}] (or1) {};
\draw (or1.input 1) -- ++(left:0.3);
\draw (or1.input 2) -- ++(left:0.3);
```



The solution in both cases is to use the intersection operator (see appendix A). First create a coordinate that specifies the horizontal position of the start of the input lines:

Absolute method:

```
\coordinate (start) at (-0.7,0);
```

The `start` coordinate's y-value is irrelevant.

Relative method:

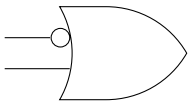
```
\draw (or1.input 1) ++(left:0.3) coordinate (start);
```

Any of the gate's input anchors could have been used.

Then draw the lines with the help of the intersection operator:

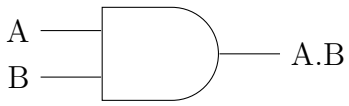
```
\draw (or1.input 1 -| start) -- (or1.input 1);
\draw (or1.input 2 -| start) -- (or1.input 2);
```

The gate would then look like this:



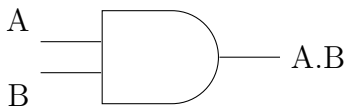
2.4 Labelling Input and Output Lines

For wide spaced inputs this looks good:



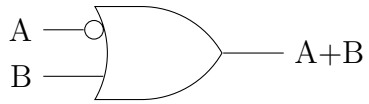

```
\draw (and1.input 1) -- node[at end,left]{A} ++(left:4mm);
\draw (and1.input 3) -- node[at end,left]{B} ++(left:4mm);
\draw (and1.output)  -- node[at end,right]{A.B} ++(right:4mm);
```

Narrow spaced inputs might need different placement:



```
\draw (and1.input 1) -- node[at end,above left]{A}
                        ++(left:4mm);
\draw (and1.input 2) -- node[at end,below left]{B}
                        ++(left:4mm);
\draw (and1.output)  -- node[at end,right]{A.B} ++(right:4mm);
```

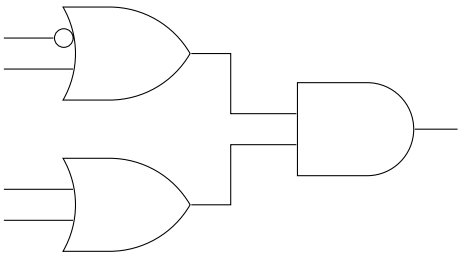
Negated inputs cause problems, but the intersection method used in §?? can be used. As an example, an absolute method with wide spaced inputs is used:




```
\coordinate (start) at (-0.7,0);
\draw (or1.input 1) -- node[at end,left]{A}
                       (or1.input 1 -| start);
\draw (or1.input 3) -- node[at end,left]{B}
                       (or1.input 3 -| start);
```

2.5 Positioning Gates

Positioning gates for straight connectors has been previously discussed (§1.2). Here the more general problem of positioning logic gates is explored. Gates can be positioned with absolute coordinates, relatively or with a matrix. A combination or at least two of these is normal. Lets see how the following diagram can be drawn using absolute and relative methods.



Absolute method

```
1  \begin{tikzpicture}[circuit logic US, scale=2]
2      \def\inx{0.25}     % input x-coord
3      \def\outx{3.25}    % output x-coord
4      \def\gxa{1}        % gate column 1
5      \def\cxa{1.75}     % connector column
6      \def\gxb{2.55}     % gate column 2
7      \def\gya{0}        % gate row 1
8      \def\gyb{-1}       % gate row 2
9      \def\gyc{-0.5}     % gate row 3
10     \node [or gate, inputs={inverted,normal}] (or1)
11           at (\gxa,\gya) {};
12     \node [or gate] (or2) at (\gxa,\gyb) {};
13     \node [and gate] (and1) at (\gxb,\gyc) {};
14     \draw (or1.input 1 -| \inx,0) -- (or1.input 1);
15     \draw (or1.input 2 -| \inx,0) -- (or1.input 2);
16     \draw (or2.input 1 -| \inx,0) -- (or2.input 1);
17     \draw (or2.input 2 -| \inx,0) -- (or2.input 2);
18     \draw (or1.output)  -- (or1.output -| \cxa,0)
19                            |- (and1.input 1);
20     \draw (or2.output)  -- (or2.output -| \cxa,0)
21                            |- (and1.input 2);
22     \draw (and1.output) -- (and1.output -| \outx,0);
23   \end{tikzpicture}
```

Lines 2–9 define macros for coordinate values, which makes adjusting the layout much easier.

Lines 10–13 draw three gates in two columns.

Lines 14–17 draw four input lines for the leftmost gates. Intersection with `(\inx,0)` is used to get a neat alignment of start points. See §?? for an equivalent method that uses a named coordinate.

Lines 18–21 draw stepped connectors from the OR gates' outputs to the AND gate's inputs. Intersection of the OR gate output's with `(\cxa,0)` is used to align the vertical parts of the connectors. See §?? for an equivalent absolute method that uses a named coordinate.

Line 22 draws an output for the left gate using path intersection with `(\outx,0)` to fix its end point.

Relative method

This is a much more compact method.

```
1  \begin{tikzpicture}[circuit logic US, scale=2,
2                      node distance=0.4]
3      \node [or gate, inputs={inverted, normal}] (or1) {};
4      \node [or gate, below=of or1 ] (or2) {};
5      \node [and gate, right=of or1, xshift=3mm,
6                                     yshift=-5mm] (and1) {};
7      \draw (or1.input 1) ++(left:3mm) coordinate (start);
8      \draw (or1.input 1 -| start) -- (or1.input 1);
9      \draw (or1.input 2 -| start) -- (or1.input 2);
10     \draw (or2.input 1 -| start) -- (or2.input 1);
11     \draw (or2.input 2 -| start) -- (or2.input 2);
12     \draw (or1.output)  -- ++(right:3mm) |- (and1.input 1);
13     \draw (or2.output)  -- ++(right:3mm) |- (and1.input 2);
14     \draw (and1.output) -- ++(right:3mm);
15 \end{tikzpicture}
```

Line 3 draws a base gate called `or1` at `(0,0)`.

Line 4 draws the next OR gate directly below this using default automatic placement. Note that the node distance has been adjusted in line 2.

Lines 5–6 draw an AND gate to the right of the OR gates. Its position is adjusted.

Line 7 creates a coordinate called `start` that is used to locate the start of the diagram's input lines

Lines 8–11 draw the OR gates input lines. The intersection operator is used to vertically align their start points.

Lines 12–13 draw stepped connectors between the OR and AND gates.

Line 14 draws the output line for the AND gate.

A Intersection Operator

To draw logic diagrams,vertical and horizontal lines have to be positioned, and connector split and jump points have to be found. TekZ's intersection operation can help with this. Consider the following code:

```
\coordinate (X) at (A -| B);
\coordinate (Y) at (A |- B);
```

This creates coordinates `X` and `Y` at the intersections of horizontal and vertical of lines extended from points `A` and `B`, as the following diagram illustrates: