

# Typesetting Karnaugh Maps with $\text{\LaTeX}$ and *TikZ*

Luis Paulo Laus

e-mail: laus@utfpr.edu.br

Version: 1.5, Version date: 2022-02-15

## Abstract

Karnaugh maps are used to simplify logic equations leading to the most compact expression of two levels for a given truth table. The drawing of them used to be a boring, annoying and error-prone task. This set of macros intend to simplify the task. They can typeset Karnaugh maps with up to twelve variables<sup>1</sup>, which is more than you might likely need<sup>2</sup>. You only have to provide a list of variable identifiers plus the truth table of your logic function. The macros also allow to highlight the simplifications of your logic function directly within a Karnaugh map. This package is based on `kvmacros.tex` from `karnaugh` package, but two drastically different ways of input the truth table are supported. The modifications carried out intended to use *TikZ* instead of native  $\text{\LaTeX}$  commands allowing easier customisation, easier extension when you need to draw other elements along with the map and resulting in higher graph quality. The labels that indicate values of the variables across de rows and columns may be represented in both numerical and simplified form.

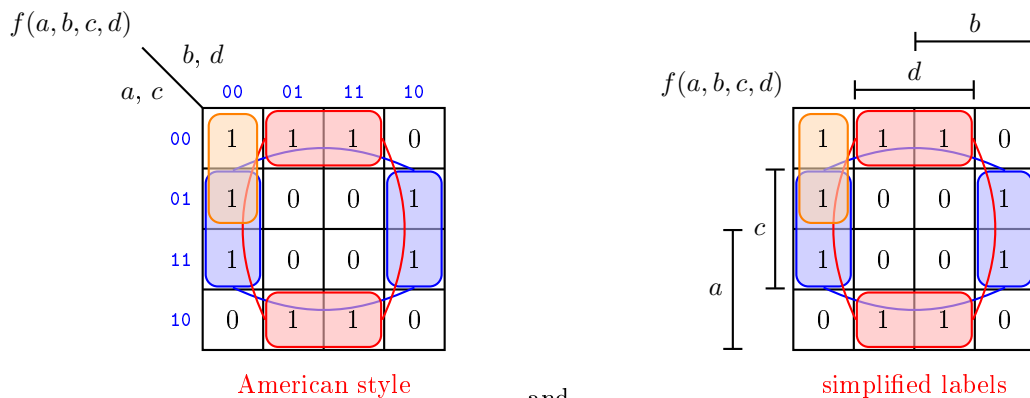
## 1 Introduction

Karnaugh maps [1] and Veitch charts are used to simplify logic equations. They are map representations of logic functions, and in that they are equivalent. Veitch charts are not supported by this package, but it should not be a big problem to port Andreas W. Wieland's `veitch` macro, available in `karnaugh` package, if you need it. Please note that this package, including its documentation, is based on Andreas W. Wieland's previous work and the author wishes to register his acknowledgment. Modifications and extensions were made in order to support customization and a diversity of input forms.

Typesetting Karnaugh maps tend to be a boring, annoying and error-prone task. Please consider using a companion free java software (JQM, see Section 8).

### 1.1 Introductory example

With `tikz-karnaugh` you can typeset big (up to twelve variables or 4096 cells) good looking maps like:



<sup>1</sup>The actual limit may be different for you.

<sup>2</sup>A twelve variables map contains of 4096 cells in a  $64 \times 64$  grid. They are simply too big to handle manually and you should consider to use a software.

On the left we have a more traditional Gray coded labels map (American style) just like the maps found in many books on Digital Electronics and Digital System Design. On the right, the same map is presented using variable bars (simplified labels). It is very easy to configure the appearance of all maps (globally) or a particular map (individually). Just change the values of some *TikZ* keys.

## 1.2 Comparison with other packages

If you ask yourself “why another Karnaugh map typesetting package?” the answer is easy: because I was not completely happy with the available packages I know and those are:

1. **karnaugh**: it is a great package that uses native  $\text{\LaTeX}$  commands to draw the map. It supports Karnaugh maps and Veitch charts. It employs a recursive algorithm with no size limit<sup>3</sup> which leads to an interesting kind of symmetry. Remember, Karnaugh maps are all about symmetry. It is not customisable, for instance, one cannot change the distance between bars<sup>4</sup> (the marks showing around the map with variable identifiers on them) and if the variable identifier is long, it will overlap another bar. Also, I want to use *TikZ* to draw colourful semi-transparent figures on top of the map to highlight groups (prime implicants) and, although it is possible, it is rather difficult and the result is not very good because they always look a bit off. I have a long-time experience with this package and I have also written a java program to draw the maps because, though typesetting simple maps is easy, highlighting the prime implicants is not.
2. **karnaughmap**: it uses *TikZ* so you got a lot of options for customisation. It is limited to eight variable which, to be honest, should be enough for anyone. The problem is that it only draws bars (those marking mentioned above) up to four variables. Also, the order in which the variables list is inputted is different from the order employed by **karnaugh**.
3. **askmaps**: this package generates configurable American style Karnaugh maps for 2, 3, 4 and 5 variables. This style is supported by **tikz-karnaugh** (see Section 6), though, in my twenty years of experience teaching the subject, I have found out that *simplified labels* are much more intuitive. Package **askmaps** contains four macros, one for each number of variables, and it can be used to highlight the prime implicants in the very same way that **karnaugh** does.
4. **karnaugh-map**: uses *TikZ* to draw up to four maps of four variables leading to a 3D six variables map. The values are input as a list of indices. It contains commands for drawing implicants on top of the map. Like **askmaps**, this package uses American style, but the label position can be centred or moved to corner (new in version 2.0).
5. **kvmap**: this is a relatively new package (released on 16 September, 2020) that allows you to typeset a Karnaugh map similarly as you use a tabular environment. The current version of **tikz-karnaugh** has a similar feature, though the syntax is somehow simpler. In Section 8 the same result is obtained using a java software, but in this case the implicants (bundles or groups) highlighting is done automatically.
6. **cartonaugh**: This is also a relatively new package (released on 15 July, 2021). It is a fork of **karnaugh-map** package that draws Karnaugh maps with up to 6 variables. The documentation of both packages looks very similar; one notable difference is the position of the function and variable labels are equivalent as used **tikz-karnaugh** if **American style** is enabled.

Roughly speaking, there are two input orders, or formats, for the function values:

- the same order as they appear on the map.
- the order they appear in a linear truth table following a canonical binary code. In some packages, namely **karnaugh-map** and **cartonaugh**, a list of indices is provided instead of an ordered list of values.

These formats must be minded very carefully in order to produce the desired map, therefore users are urged to read Section 1.3 and, for more details, Section 1.4. In Section 8 a companion software that helps with inputting the data necessary to produced the map is discussed.

Also, there are two main styles:

<sup>3</sup>It works until you blow the memory out which will happen about ten to twelve variables.

<sup>4</sup>Those bars have been underappreciated along the history. Karnaugh [1] himself called them “simplified labels” and used them only to replace the Gray coded numbers showing around the map. Their true strength is the ease way they point out which variable belong to a prime implicant and which does not. An approach much easier than interpreting the Gray coded numbers.

- American style is very popular in books, the association of variable and cell position is shown by Gray coded number outside the map grid;
- simplified labels represent graphically the regions (columns or rows) for which a given variable is true or false. Because Karnaugh map is a graphical method, this style is very appealing although much less popular than American style.

The following table shows how each package deals with those two features.

Package	Input Order		Style	
	Truth Table	Karnaugh Map	American	Simplified Labels
karnaugh	✓			✓
karnaughmap	✓		✓	✓
askmaps	✓		✓	
karnaugh-map	✓*		✓	
kvmap		✓	✓	
cartonaugh	✓*		✓	
tikz-karnaugh	✓	✓	✓	✓

\* List of indices (cell position) instead list of values.

The only package that supports both input orders or formats is `tikz-karnaugh`. With `tikz-karnaugh` you can typeset big (up to twelve variables or 4096 cells) good looking maps with Gray code labels (American style) or bars (simplified labels). Using a companion free java software (JQM, see Section 8), you can do it automatically, including highlighting the solution.

### 1.3 Inputting the truth table

To create a Karnaugh map, the first thing you have to do is to load `TikZ`. For this, type `\usepackage{tikz}` in the preamble of your document. Then, if the package is somewhere `TeX` can find it, load the library with the command `\usetikzlibrary{karnaugh}`. If it is not, you can use something like `\input tikzlibrarykarnaugh.code`. You may need to provide the full or relative path to file `tikzlibrarykarnaugh.code.tex`. User of online services like overleaf (<https://www.overleaf.com/>) will need to check to version and, if it is not updated, will also need to upload the file `tikzlibrarykarnaugh.code.tex` to same folder.

Now, consider the logic function  $f(a, b, c, d)$  of Section 1.1. If you need to typeset a Karnaugh map for  $f(a, b, c, d)$ , it is likely that this function would be given in one of two ways: 1) a truth table; 2) a Karnaugh map (from a book, for example). Returning to Section 1.1, the most important data are the values that function assumes. This can be represented in a chart like:

1	1	1	0
1	0	0	1
1	0	0	1
0	1	1	0

Other relevant information is the set of input variables and their relation with rows and columns of the map:  $a$ ,  $c$ ,  $b$  and  $d$ , observe the order. So, the example of Section 1.1 can easily be put into a Karnaugh map by using the `\karnaughmaptab` macro in a `TikZ` environment (`\begin{tikzpicture}`) or inline command (`\tikz`) as:

$f(a,b,c,d)$

	b			
	d			
	1	1	1	0
c	1	0	0	1
	1	0	0	1
a	0	1	1	0

```
\tikz[karnaugh]{
  \karnaughmaptab{4}{f(a,b,c,d)}{acbd}{%
    1110
    1001
    1001
    0110
  }{}
}
```

Inline command `\tikz` receives a list of options and one mandatory parameter, the macro `\karnaughmaptab` itself. The `\karnaughmaptab` macro has five mandatory parameters:

1. the number of variables in the map, 4 in this example;
2. an identifier for the function,  $f(a,b,c,d)$  in this example;
3. a list of variable identifiers for the variables,  $acbd$  in this example (order is important);
4. the list of values of  $f(a,b,c,d)$  as they appear in the map<sup>5</sup>; and
5. a possibly empty set of TikZ commands that will be drawn before the function values so the values will appear on top of them.

The variable identifiers in the third parameter are ordered such that the variables associated with rows appear before the variables associated with columns and the most significant variable for the Gray code outside the map appear first. In other words, list the variables on left side of the map from most to least significant and then list the variables on top of the map also from most to least significant.

The fifth parameter remains empty in this example, it will be discussed further on.

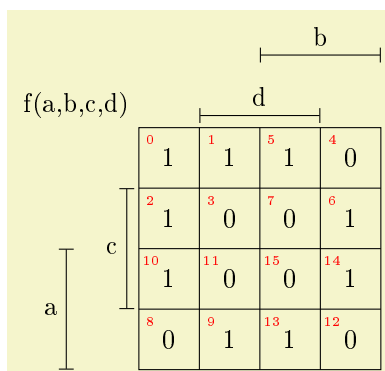
Suppose now, instead of a Karnaugh map, you have a logic function  $f(a,b,c,d)$  with the following truth table:

---

<sup>5</sup>Spaces and newlines are not important and can be added for the sake of readability. If, however, you need to skip one cell, insert a pair of curly brackets, `{}`. It is usually a good idea to replace all zeros (or ones if you desire a zero map) by pairs of curly brackets. Karnaugh map is a graphical method and overloading the map with useless information can undermine the user ability to find the solution.

Index	$a$	$b$	$c$	$d$	$f$
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

Rewriting the function values in the correct order as they should appear in the map is time consuming and error-prone. So, why not let L<sup>A</sup>T<sub>E</sub>X take care of the details for you? This logic function can be put into a Karnaugh map by using the `\karnaughmap` macro in a TikZ environment (`\begin{tikzpicture}`) or inline command (`\tikz`) as:



```
\tikz[karnaugh, enable indices]{
  \karnaughmap{4}{f(a,b,c,d)}{abcd}{%
    1110 0110 0110 0110
  }{}
}
```

Note that, this time, the order in which parameters #3 and #4 are typed in is different from the previous example. Parameter #3 is the list of input variables in the most to least significant bit as in truth table. Parameter #4 is just the last columns of truth table. The indices (little red number inside a cell) are enabled so one can relate each cell on the map to its position in the truth table. In other word, the index in the cell corresponds to the index in truth table.

The `\karnaughmap` macro, just like `\karnaughmaptab`, has five mandatory parameters:

1. the number of variables in the map;
2. an identifier for the function;
3. a list of variable identifiers for the variables, abcd (compare with the last example);
4. the list of values of  $f$  for each line in the truth table; and
5. a possibly empty set of TikZ commands that will be drawn before the function values so the values will appear on top of them.

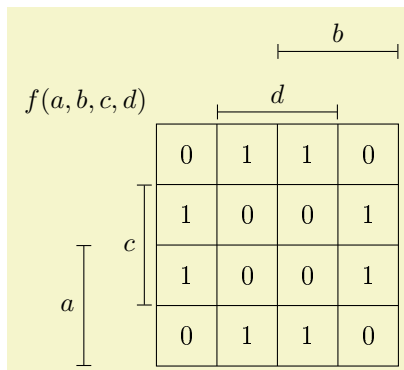
The variable identifiers in the third parameter are ordered from highest to lowest significance (the same way as in the truth table, with  $a$  having a significance of  $2^3 = 8$  and  $d$  having a significance of  $2^0 = 1$ ). The list of values of  $f$  was read from lowest to highest index. The fifth parameter remains empty in this example, it will be discussed further on.

The indices in the upper left corner of each cell corresponds to the indices in the truth table. The indices can easily be calculated from the variable value in the truth table, e.g., the index for row 11 is given by:

$$2^3 a + 2^2 b + 2^1 c + 2^0 d = 8a + 4b + 2c + 1d = 8 + 2 + 1 = 11.$$

The macros that read the variables list and the list of logic values (i.e., parameters #3 and #4) work recursively. This is why variables  $a$  and  $c$  are assigned to Karnaugh map rows and  $b$  and  $d$  to columns. If you desire a different variable arrangement, you will not only need to change parameter #3, but also #4 accordingly. This can be troublesome; see Section 8 for a java software that can help in this matter.

Each entry has to be one character long and spaces are allowed<sup>6</sup>, otherwise – like a variable identifier enclosed in  $\$$ s – you have to put it into curly brackets:



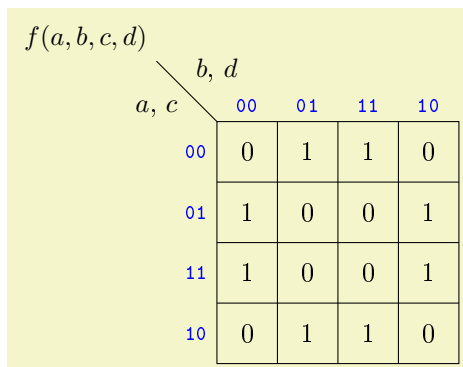
```

\begin{tikzpicture}[karnaugh]
  \karnaughmap{4}{f(a,b,c,d)}{{a$}{b$}{c$}{d$}}%
    {0110 0110 0110 0110}{}
\end{tikzpicture}

```

Observe that the labels are all in math mode in this example. Also, a TikZ environment was used so `\karnaughmap` macro is not into curly brackets which is mandatory when using `\tikz` because `\karnaughmap` creates several path commands. Moreover, the indices were omitted by removing `enable indices` from the options list.

If you prefer Gray coded labels, referred herein as *American style*, just type `American style` in the option list:



```

\begin{tikzpicture}[karnaugh, American style]
  \karnaughmap{4}{f(a,b,c,d)}{{a$}{b$}{c$}{d$}}%
    {0110 0110 0110 0110}{}
\end{tikzpicture}

```

See Section 6 for more details on American styled maps.

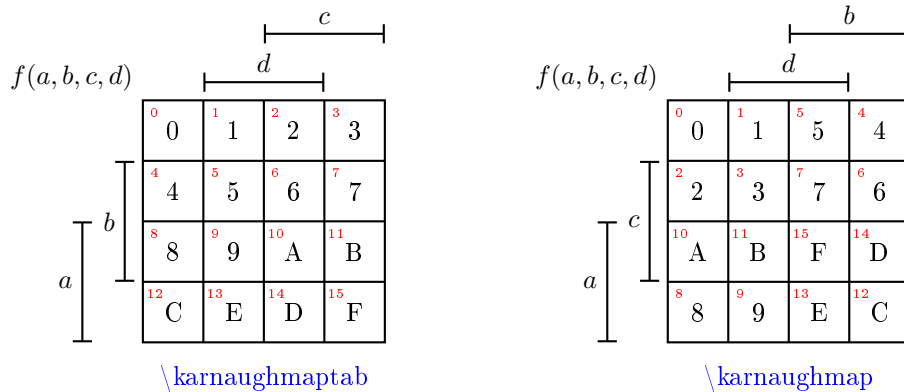
## 1.4 Comparison between `\karnaughmaptab` and `\karnaughmap`

Before we go any further in the details, it is important to understand the difference between the two input modes: truth table and table formatted Karnaugh map. If the input data is given in a linear format where the binary code associated with the input variables imposes the natural order in which the output function is written, `\karnaughmap` must be used to create a Karnaugh map in the appropriated fashion. Moreover, the input variables list also has to be written from the more to least significant bit that imposes that order. If, however, you already have a Karnaugh map and want to typeset it, it is usually much easier to type the output values in the order they appear in the Karnaugh map just like it was a simple *table* using

<sup>6</sup>White spaces are really usable to make the string more readable leading to fast verification.

`\karnaughmaptab`. The weaky analogy between tabular environment gives the `tab` at the end of the macro name. Moreover, the input variables list has to be written in the order they appear on the given map.

Let's see how it works calling both macros with the same parameters. Instead of zeros, ones, white spaces and don't-cares, let's use a string of hexadecimal numbers from 0 to F, so we can track where the symbols go in the map. The indices are enabled, so you can see the values go in the correct position, just the notion of "correct" changes from map to map according to the macro used. Bellow, on the left we have `\karnaughmaptab` and the right `\karnaughmap`.



Both macros were called with `{4}{f(a,b,c,d)}{a}{b}{c}{d}{0123 4567 89AB CEDF}`, but the result is different because each macro treats its arguments in a different way. `\karnaughmaptab` uses the same order the values are given, just put the value in the correct cell automatically changing the row when needed. The input variables also follow in the given order: first the variable at left side then at the top of the map. `\karnaughmap` on the other hand does something much more sophisticated. It sorts the values such that they fall in the correct cell following the given linear order. It also places the input variables in the correct position according to the same order. Now, for four variables, there are 24 ways to assign the variables to a map. `\karnaughmap` takes the last given input variable and assigns it to the least significant bit associated to the columns. The second last is assigned to the least significant bit associated to the rows. The next from the right to left is again assigned to column, and that goes on until the first given input variable is assigned. The corresponding cell placement is done by a recursive macro. If you are not satisfied with the default assignment you can use a java software to customize the map any way you like (see Section 8).

## 2 Karnaugh Map Library

### TikZ Library `karnaugh`

```
\usepgflibrary{karnaugh} % ETeX and plain TeX and pure pgf
\usepgflibrary[karnaugh] % ConTeXt and pure pgf
\usetikzlibrary{karnaugh} % ETeX and plain TeX when using TikZ
\usetikzlibrary[karnaugh] % ConTeXt when using TikZ
```

This library provides TeX macros to typeset Karnaugh maps. This library defines the following key:

`/tikz/karnaugh` (no value)

This key should be passed as an option to a picture or a scope that contains a map, i.e., that calls `\karnaughmap` or `\karnaughmapvert` macros. It will do some internal setups.

`\karnaughmap`{*num var*}{*function*}{*var list*}{*contents*}{*decoration*}

This macro creates a Karnaugh map of *num var* variables for variable *function* as a function of the variables listed in *var list* for the values given in *content* and applying the specified *decoration*. Any but the first parameter can be empty.

`\karnaughmaptab`{*num var*}{*function*}{*var list*}{*contents*}{*decoration*}

Similar to `\karnaughmap`, but the order in which the *var list* and *content* are given correspond to the map it creates.

`\karnaughmapvert`{*num var*}{*function*}{*var list*}{*contents*}{*decoration*}

Similar to `\karnaughmap`, but map will be transposed (like in matrix transposition). See Section 7 for more details.

`\karnaughmaptabvert`{*num var*}{*function*}{*var list*}{*contents*}{*decoration*}

Similar to `\karnaughmaptab`, but map will be transposed (like in matrix transposition).

`\pgfmathdectoGray`{*macro*}{*number*}

Defines *macro* as the result of converting *number* from base 10 to Gray coded binary. The idea is to provide a new macro compatible with other base conversion macros described in Section 95.4 Base Conversion of reference [2].

```
1001 \pgfmathdectoGray\mynumber{14}\mynumber
```

`\kmdectobin`{*number*}

Converts *number* from base 10 to binary with the number of digits equal to `\kmvarno`, the number of variables in the map.

```
001001 \kmvarno=6\relax\kmdectobin{9}
```

`\kmdectoKG`{*number*}

Converts *number* (usually the cell index) to a binary code that resembles Gray code deinterleaving the variables with the number of digits equal to `\kmvarno`, the number of variables in the map. See `kmcell/.style` for an example of application.

```
010001 \kmvarno=6\relax\kmdectoKG{9}
```

`\kmdectoKGdec`{*number*}

Converts *number* (usually the cell index) to a decimal number equivalent to the binary code that resembles Gray code deinterleaving the variables. It uses `\kmxsize` to compute the correct amount. This macro is meant for tests purposes, but it can help advanced users to understand how a map is assembled. See Section 8 for an example of use.

```
11 \kmxsize=4\relax\kmdectoKGdec{13}
```

`/tikz/every karnaugh`

(style, initially empty)

The style automatically applied to every Karnaugh map. It can be globally set using `\tikzset` and this can be important to standardizing your maps along the document.

`\kmindexcounter`

A  $\TeX$  counter for cell index. See `kmcell/.style` for an example of application.

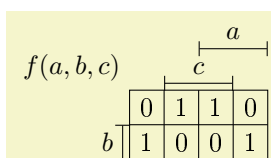
`\kmunitlength`=*{length}*}

This length sets the size of an individual cell in the map. It can be set by `karnaugh cell size`. It is used to keep dimensions proportional.

`/tikz/karnaugh cell size`=*{dimension}*

(no default, initially 8mm)

Sets the cell size and, consequently, all other dimensions that depends on the cell size. It sets the length `\kmunitlength` which is used internally but can also be used by the user when needed. Because `karnaugh` uses `\kmunitlength` to set a lot of things, it is a good idea to set the cell size first.



```
\begin{tikzpicture}[karnaugh cell size=1.3em, karnaugh]
\karnaughmap{3}{f(a,b,c)}{{a$}{b$}{c$}}{0110 0110}{}
\end{tikzpicture}
```



Care should be taken because not all dimensions depend on the cell size, for example, the arrow tip width of variable bars does not, so the arrow tip may touch the map if `kmbar sep` is not set to a suitable value.

`/tikz/American style=<boolean>` (default true, initially false)

Boolean switch that changes the map layout to American style. The initial value is `false` meaning that the simplified labels (bars with a variable identifier on it) will be drawn unless they are explicitly replaced by American style (Gray coded labels). See Section 6 for details.

$f(a,b,c)$		$a, c$			
		$00$	$01$	$11$	$10$
$b$	$0$	0	1	1	0
	$1$	1	0	0	1

```
\begin{tikzpicture}[karnaugh, American style]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/disable bars=<boolean>` (default true, initially false)

Boolean switch that disables the typesetting of all bars and the function identifier. The original intention was to allow for manually crafted American style maps, but since version 1.3 this map style is fully supported so this switch lost its purpose. The initial value is `false` meaning that the bars will be typeset unless they are explicitly disabled.

0	1	1	0
1	0	0	1

```
\begin{tikzpicture}[karnaugh,disable bars]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

Note that `$f(a,b,c)$` and `{{a}{b}{c}}` are not used and could be empty in the example above.

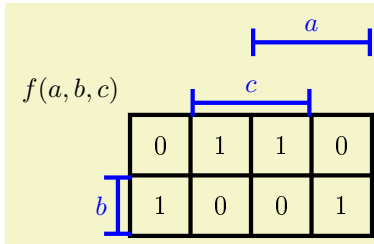
`/tikz/kmbar` (style, initially |-)

The style used for the top and side bars related to the variables and denoting the rows and columns for which the respective variable is 1. The initial value is `|-|` meaning they all will be represented as a line with T shaped tips.

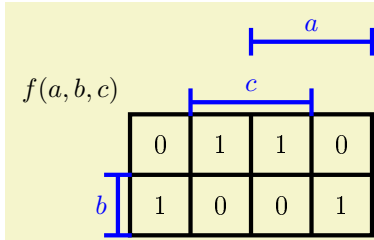
$f(a,b,c)$		$a$			
		$00$	$01$	$11$	$10$
$b$	$0$	0	1	1	0
	$1$	1	0	0	1

```
\begin{tikzpicture}[karnaugh,
  kmbar/.style={blue,<->}]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

See Chapter 16 Arrows of reference [2] for more arrow options and proper control over arrow appearance. For instance, if you want to draw a bar that ends exactly aligned with the lines inside the map, you can change the style to `|-|,shorten >=-0.2pt,shorten <=-0.2pt`, assuming you are using the default thickness `thin` which implies `line width=0.4pt` (refer to Section 15.3.1 Graphic Parameters: Line Width, Line Cap, and Line Join of reference [2] for *TikZ* line width). Otherwise, the very end of the line tip will be aligned with the middle of the internal lines, but someone should have to be very careful to notice a difference of `0.2pt`. Compare the two examples below where `ultra thick` is used for emphasis.



```
% sloppy alignment
\begin{tikzpicture}[karnaugh, ultra thick,
  kmbar/.style={blue,|-|}]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

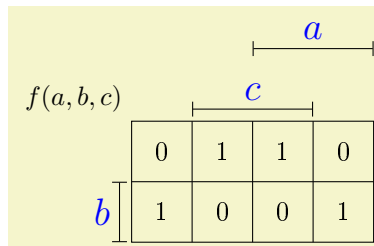


```
% exact alignment
\begin{tikzpicture}[karnaugh, ultra thick,
  kmbar/.style={blue,
  |-,
  shorten >=-0.8pt,
  shorten <=-0.8pt}]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmbar label`

(style, initially empty)

The style used for the variable identifiers on the bars. For American styled maps, it means the style for typesetting the lists of variables associated with columns and rows.

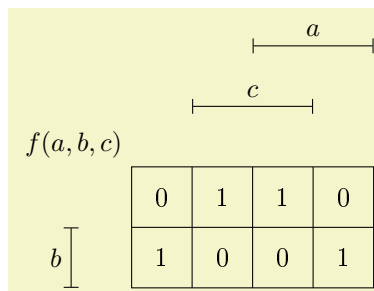


```
\begin{tikzpicture}[karnaugh,
  kmbar label/.style={blue,font=\Large}]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmbar sep=<width>`

(no default, initially 0.2\kernlength)

The distance between the bar closer to the map and the map itself. It depends mainly on the line tip used in `kmbar/.style` and the thickness of the map set by `kmbox/.style`.

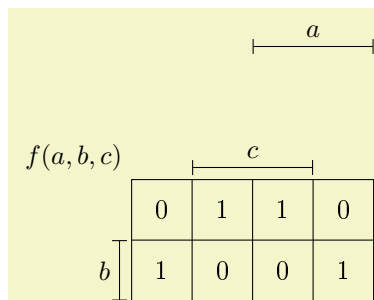


```
\begin{tikzpicture}[karnaugh,
  kmbar sep=1\kernlength]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmbar top sep=<width>`

(no default, initially 1\kernlength)

The distance between two bars on top of map. It depends mainly on the font height used in `kmbar label/.style`.

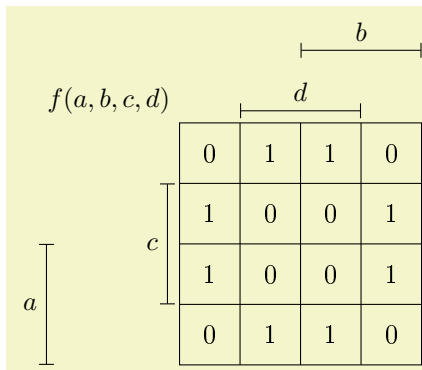


```
\begin{tikzpicture}[karnaugh,
  kmbar top sep=2\kernlength]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmbar left sep=<width>`

(no default, initially  $1\text{\kernunitlength}$ )

The distance between two bars at the left side of map. It depends mainly on the variable identifier width and the font size used in `kmbar label/.style`.

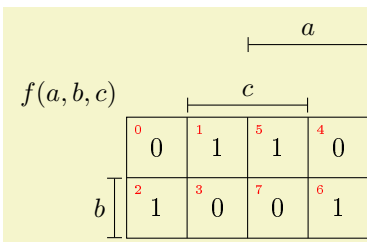


```
\begin{tikzpicture}[karnaugh,
                    kmbar left sep=2\kernunitlength]
\karnaughmap{4}{f(a,b,c,d)}{{a}{b}{c}{d}}%
{0110 0110 0110 0110}{}
\end{tikzpicture}
```

`/tikz/enable indices=<boolean>`

(default true, initially false)

Boolean switch that enables the typesetting of all indices. The index corresponds to the row number in the truth table respective to the cell. The style `kmindex/.style` can modify how the indices are presented and the lengths `kmindex posx` and `kmindex posy` where they are placed. Base-10 is the default base for indices, but there are two Boolean switches, `binary index` and `Gray index`, to change the encoding. The initial value is `false` meaning that the indices will not be typeset unless they are explicitly enabled.

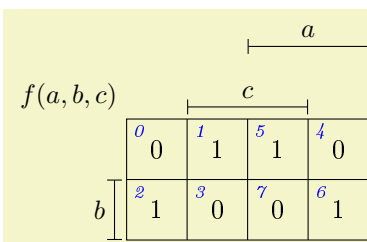


```
\begin{tikzpicture}[karnaugh,
                    enable indices]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmindex`

(style, initially `red,font=\tiny`)

The style used for cell index if enable (see also `enable indices`).

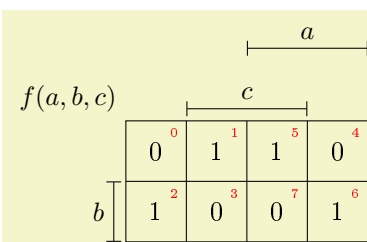


```
\begin{tikzpicture}[karnaugh,
                    enable indices,
                    kmindex/.style={blue,
                                     font=\scriptsize\itshape}]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmindex posx=<dimension>`

(no default, initially  $0.2\text{\kernunitlength}$ )

The horizontal distance from the cell left side to the index centre.

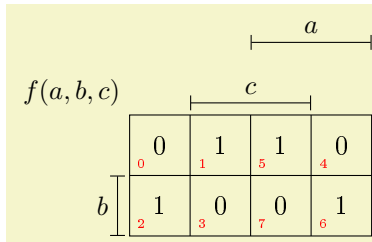


```
\begin{tikzpicture}[karnaugh,
                    enable indices,
                    kmindex posx=.8\kernunitlength]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmindex posy=<dimension>`

(no default, initially  $0.8\text{\kernunitlength}$ )

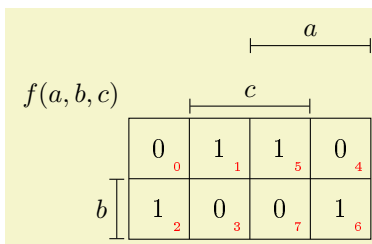
The vertical distance from the cell bottom to the index centre.



```
\begin{tikzpicture} [karnaugh,
  enable indices,
  kmindex posy=.2\kernunitlength]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmindex pos={x coordinate}{y coordinate}`

Sets `kmindex posx` and `kmindex posy` to x and y coordinates measured in `\kernunitlength` from the cell bottom left corner.

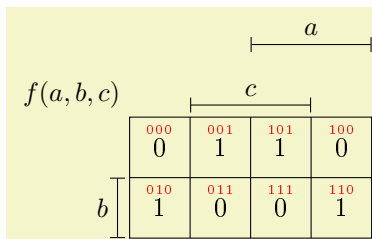


```
\begin{tikzpicture} [karnaugh,
  enable indices,
  kmindex pos={0.8}{0.2}]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/binary index=<boolean>`

(default true, initially false)

Boolean switch that sets the index presentation to binary code. It is convenient to also set the index coordinates. In the following example, the significance order is *a*, *b* and *c*, meaning, *a* is the most significant bit and *c* is the least significant bit. Therefore, the left most bit of the indices is one only in the two left columns below *a* bar, the middle bit is one in the bottom row as *b* bar extends and the right most bit is one in the central columns below *c* bar.



```
\begin{tikzpicture} [karnaugh,
  enable indices,
  binary index,
  kmindex pos={0.5}{0.8}]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/Gray index=<boolean>`

(default true, initially false)

Boolean switch that sets the index presentation to binary code corresponding to the aggregate value of the variables for one cell and not the index of the truth table row respective to the cell. The difference between `Gray index` and `binary index` is that the latter is a binary coded representation of the cell position in the truth table and, because the recursive algorithm that builds the map scrambles (interleaves) the variables, it looks random at first sight. The former, descrambles (deinterleaves) the variables so it would look like a string of zeros and ones associated with row variables followed by one associated with columns in the order they are shown along the map and not in macro parameter #3. It looks like Gray code, that's where its name came from, but in fact it is not. Therefore, `Gray index` is pretty much useless to locate the corresponding row in the truth table. However, it can be used to visualize the state of variables more easily than binary code. A secondary index can be set, see `kmcell/.style` for an example of how it can be done. Normally, enabling binary or Gray coded indices requires that the index position be adjusted. The initial value of this switch is `false` meaning that the indices will be typeset accordingly to another code, decimal or binary, depending on `binary index` switch.

$f(a, b, c)$

		$a, c$			
	$b$	00	01	11	10
0		<sup>000</sup> 0	<sup>001</sup> 1	<sup>011</sup> 1	<sup>010</sup> 0
1		<sup>100</sup> 1	<sup>101</sup> 0	<sup>111</sup> 0	<sup>110</sup> 1

```
\begin{tikzpicture}[karnaugh,
    American style,
    enable indices,
    Gray index,
    kmindex pos={0.5}{0.8}]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmsep line`

(style, initially empty)

The style applied to the line that separates the list of variables assigned to columns from the list of variables assigned to rows in an American styled map.

$f(a, b, c)$

		$a, c$			
	$b$	00	01	11	10
0		0	1	1	0
1		1	0	0	1

```
\begin{tikzpicture}[karnaugh,
    American style,
    kmsep line/.style={thick,draw=blue}]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmsep line length= $\langle dimension \rangle$`

(no default, initially  $1\text{kmunitlength}$ )

The length divided by  $\sqrt{2}$  of the line that separates the list of variables assigned to columns from the list of variables assigned to rows in an American styled map. Actually, this key sets the vertical and horizontal distances from the top left of the map to the end of the line. The initial value of  $1\text{kmunitlength}$  means that the line has a total length of  $\sqrt{2}\text{kmunitlength}$ .

$f(a, b, c)$

		$a, c$			
	$b$	00	01	11	10
0		0	1	1	0
1		1	0	0	1

```
\begin{tikzpicture}[karnaugh,
    American style,
    kmsep line/.style={thick,draw=red},
    kmsep line length=0.5kmunitlength]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmcell`

(style, initially empty)

The style used for cell contents.

$f(a, b, c)$

		$a$			
		$c$			
	$b$	0	1	1	0
		1	0	0	1

```
\begin{tikzpicture}[karnaugh,
    kmcell/.style={blue,font=|Large}]
\karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

Some interesting applications of `kmcell/.style` involves the cell index given by `\the\kmindexcounter`. You can name every cell for future use or place a label within the cell index just like `enable indices` does. In the following example, `kmcell/.style` is used to place a label within each cell with the decimal value of the cell index and `enable indices` is for the binary value. Moreover, the cell content is also a number that corresponds to the cell index (manually placed) just to show the correlation, namely, the cell index corresponds to the cell position it the parameter #4.

		a			
		c			
b	0	1	5	4	
	2	3	7	6	
	000 0	001 1	101 5	100 4	
	010 2	011 3	111 7	110 6	

```
\begin{tikzpicture} [karnaugh,
enable indices,
binary index,
kmindex pos={0.5}{0.8},
kmc cell/.style={green!60!black,
label={{font=\scriptsize,blue,
label distance=-0.3\kmunittlength}
below left:|the\kmindexcounter}}}]
\karnaughmap{3}{f(a,b,c)}{{a$}{b$}{c$}}{0123 4567}{
\end{tikzpicture}
```

So, let's compare the binary index, internally generated by `\kmdectobin`, with pseudo index generated by `\kmdectoKG` when Gray index is set:

		a			
		c			
b	0	1	5	4	
	2	3	7	6	
	000 0 000	001 1 001	101 5 011	100 4 010	
	010 2	011 3	111 7	110 6	
	100	101	111	110	

```
\begin{tikzpicture} [karnaugh,
enable indices,
binary index,
kmindex pos={0.5}{0.8},
kmc cell/.style={green!60!black,
label={{font=\tiny,blue,
label distance=-0.2\kmunittlength}
below:|kmdectoKG{\kmindexcounter}}}]
\karnaughmap{3}{f(a,b,c)}{{a$}{b$}{c$}}{0123 4567}{
\end{tikzpicture}
```

The binary index at the top in red is the representation of the cell position in binary so it contains the values of *abc*. Pseudo Gray code at the bottom in blue is the deinterleaved values of the cell position placing *b* in the leftmost position, in other words, it is just *bac*. This code is convenient to represent the code obtained placing the variables side by side as they appear in the left and top of the map. Curious readers are invited to relate the values indicated by the variable bars with the red and blue codes.

`/tikz/kmlabel top` (style, initially above,blue,font=\footnotesize\ttfamily)

The style used for Gray coded labels at the top of American styled maps. Variable `\x` can be used to express the column position from 0 to `\kmsize-1`. The distance from the map can be controlled through `/tikz/yshift=(dimension)`, `/pgf/inner sep=(dimension)`, `/pgf/inner ysep=(dimension)`, `/pgf/minimum width=(dimension)` or `above=(dimension)`, for instance. Naturally, `yshift`, `inner ysep` and `above` depend on the rotation and might need to be replaced by something else like `xshift`, `inner xsep` or `right`, as in the example below.

		a, c			
		00	01	11	10
b	0	0	1	1	0
	1	1	0	0	1

```
\begin{tikzpicture} [karnaugh,
American style,
kmlabel top/.style={red,font=\small,
rotate=90,right=4pt,
draw=cyan,very thin,
label={{green!60!black,
label distance=1em}right:|x}}]
\karnaughmap{3}{f(a,b,c)}{{a$}{b$}{c$}}%
{0110 0110}{
\end{tikzpicture}
```

`/tikz/kmlabel left` (style, initially left,blue,font=\footnotesize\ttfamily)

The style used for Gray coded labels at the left side of American styled maps. Variable `\y` can be used to express the row position from 0 to `\kmsize-1`. The distance from the map can be controlled through `/tikz/xshift=(dimension)`, `/pgf/inner sep=(dimension)`, `/pgf/inner xsep=(dimension)`, `/pgf/minimum width=(dimension)` or `left=(dimension)`.

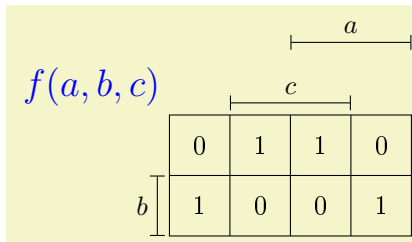
		a, c			
		00	01	11	10
b	0	0	1	1	0
	1	1	0	0	1

```
\begin{tikzpicture} [karnaugh,
American style,
kmlabel left/.style={red,font=\small,
left=4pt,draw=cyan,very thin,
label={{green!60!black}left:|y}}]
\karnaughmap{3}{f(a,b,c)}{{a$}{b$}{c$}}%
{0110 0110}{
\end{tikzpicture}
```

`/tikz/kmvar`

(style, initially empty)

The style used for the variable name (function) of the map.

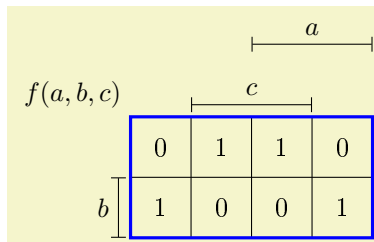


```
\begin{tikzpicture}[karnaugh,
                    kmvar/.style={blue,font=|Large|}]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmbox`

(style, initially empty)

The style used for the box surrounding the map.

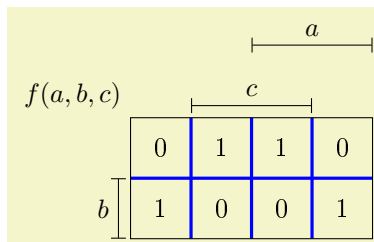


```
\begin{tikzpicture}[karnaugh,
                    kmbox/.style={blue,very thick}]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

`/tikz/kmlines`

(style, initially empty)

The style used for the lines separating adjacent rows and columns inside the map.

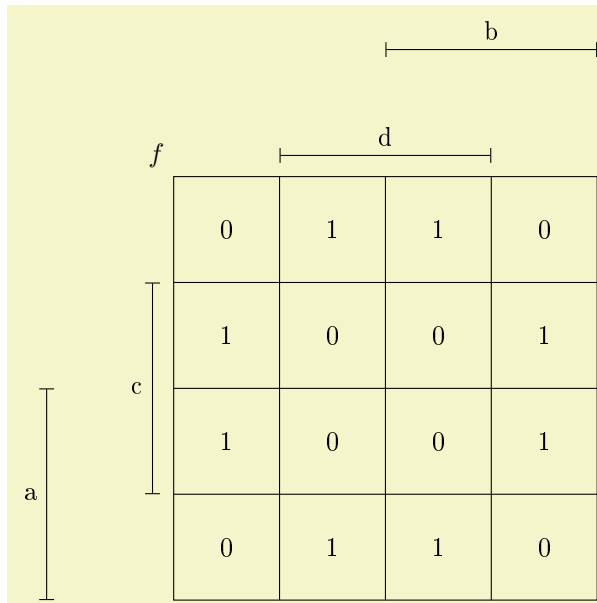


```
\begin{tikzpicture}[karnaugh,
                    kmlines/.style={blue,very thick}]
  \karnaughmap{3}{f(a,b,c)}{{a}{b}{c}}{0110 0110}{}
\end{tikzpicture}
```

You can add options to the graphics by setting the `every karnaugh/.style` which is automatically applied.

### 3 Adjusting the map size

Possibly the most important feature that you can change is the size of the diagrams and it is done by changing the size of the cells within the map, simply by typing:

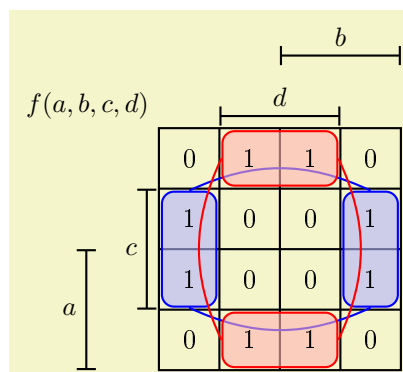


```
\begin{tikzpicture}[karnaugh cell size=14mm,
karnaugh]
\karnaughmap{4}{f}{abcd}{0110 0110 0110 0110}{}
\end{tikzpicture}
```

You can set the cell size using `karnaugh cell size` locally or globally using `\tikzset{karnaugh cell size=14mm}` or modifying the length `\kmunitlength`. The setting of the `\kmunitlength` remains active until you change it again<sup>7</sup>. Just remember that, if you need to define the coordinate multiple (x or y) in terms of `\kmunitlength`, you must set `karnaugh cell size` *before* setting x or y.

## 4 Marking simplifications

The already mentioned fifth parameter can be used if you want to draw something inside the Karnaugh map. For example, this is useful if you want to show how you simplified a logic function highlighting the prime implicants:



```
\begin{tikzpicture}[karnaugh,
thick,
grp/.style n args={3}{#1,fill=#1!30,
minimum width=#2\kmunitlength,
minimum height=#3\kmunitlength,
rounded corners=0.2\kmunitlength,
fill opacity=0.6,
rectangle,draw}]
\karnaughmap{4}{f(a,b,c,d)}{{a$}{b$}{c$}{d$}}%
{0110 0110 0110 0110}%
{
\node[grp={blue}{0.9}{1.9}](n000) at (0.5,2.0) {};
\node[grp={blue}{0.9}{1.9}](n001) at (3.5,2.0) {};
\draw[blue] (n000.north) to [bend left=25] (n001.north)
(n000.south) to [bend right=25] (n001.south);
\node[grp={red}{1.9}{0.9}](n100) at (2.0,3.5) {};
\node[grp={red}{1.9}{0.9}](n110) at (2.0,0.5) {};
\draw[red] (n100.west) to [bend right=25] (n110.west)
(n100.east) to [bend left=25] (n110.east);
}
\end{tikzpicture}
```

and the corresponding expression is:

$$f(a,b,c,d) = c\bar{d} + \bar{c}d$$

where colours were used to relate the subexpression with the prime implicant highlighted on the map.

Instead of L<sup>A</sup>T<sub>E</sub>X's graphics macros, as in the original package, you can use TikZ for this purpose. In this example, a new style `grp` was defined in order to draw semi-transparent rectangles with a specified colour, width and height (both given in `\kmunitlength`). The Karnaugh map has its datum at the lower left point *exactly*. The centre point coordinates of those rectangles are specified using the `at` command.

<sup>7</sup>Or, of course, until you leave the group in which you redefined the value.



The length of a single cell within the Karnaugh map is equal to `\kmunitlength`. Thus, the `x` and `y` units are set to `1\kmunitlength` by the `tikzpicture` environment option `karnaugh` so the coordinates can be written without the unit and the rectangles will fall in the precise position even if one changes the map size by changing the `\kmunitlength`. Just remember that if `\kmunitlength` is set, using perhaps `karnaugh cell size`, *after* the option `karnaugh` you will need to set `x` and `y` manually. Thus, it is preferable to set `\kmunitlength` *before* option `karnaugh`. As a good design practice, always place `karnaugh cell size` first and then write `karnaugh` in the option list when you need a custom size.

Highlighting the solution can be an arduous and error-prone task. See Section 8 for JQM, a java free software that can generate Karnaugh maps with the implicants automatically highlighted and the simplified equation colour coded accordingly to the highlights. You will still need `tikz-karnaugh` package to render the map. By the way, the highlights seen in this section and the next two sections were generated using JQM.

## 5 Complete example

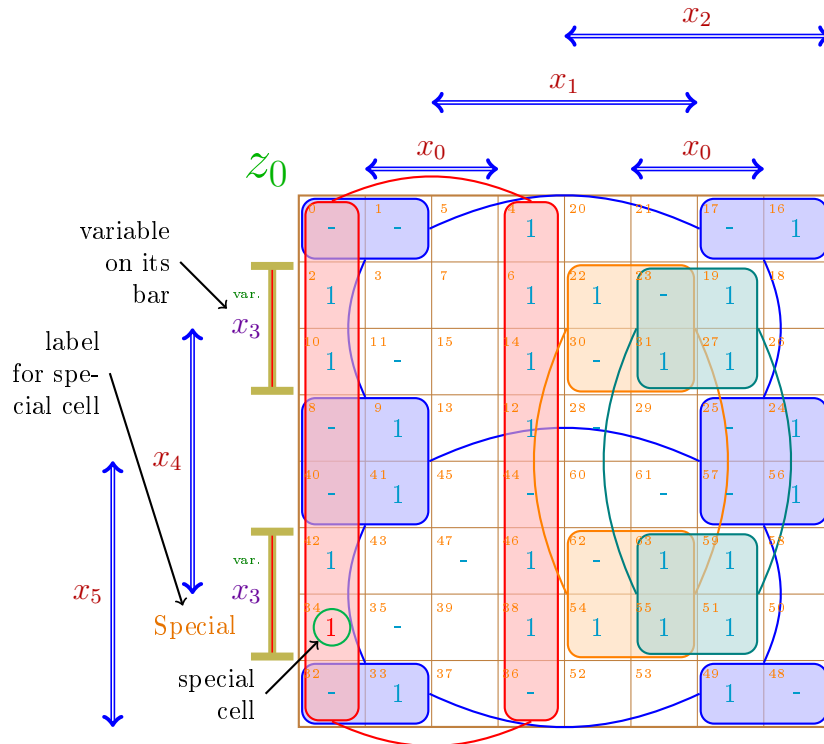
In this and in the next sections, examples of how individual variables and cell contents can be format are presented. The syntax relies on square brackets (`[]`) to enclose TikZ features that change the appearance and add more graphs to an individual variable or cell content. Let us see a more interesting and colourful example:

```

\begin{tikzpicture}[karnaugh cell size=2.5em, karnaugh,
  thick,
  grp/.style n args={3}{#1,fill=#1!30,
  minimum width=#2|kmunitlength,
  minimum height=#3|kmunitlength,
  rounded corners=0.2|kmunitlength,
  fill opacity=0.6,
  rectangle,draw},
  kmbar/.style={blue,<->,double=white,semithick},
  kmbar left sep=1.2|kmunitlength,
  kmbar sep=0.4|kmunitlength,
  kmbar label/.style={red!70!black,font=|large},
  kmindex/.style={orange,font=|tiny},
  enable indices,
  kmcell/.style={cyan!80!black},
  kmbox/.style={brown,thick},
  kmllines/.style={brown,thin},
  kmvar/.style={green!70!black,font=|huge},
  lbl/.style={left,align=right,text width=1.5|kmunitlength}]
\karnaughmap{6}{z_0}{x_1}{x_2}{x_3}{x_4}{x_5}{x_6}
{[yellow!70!black,name=Nv,|-|,double=red,very thick,
  label={font=|tiny,green!50!black}above:var.},
  text=blue!60!red]x_3}
{x_0}
{-1{}1{}1{}1{}-11-1{}1{}1{}-{}1{}1{}1-1-{}1-{}-1-1%
  {[red,name=Nc,label={name=Nl,orange!90!black,
  label distance=1|kmunitlength}left:Special},
  circle,inner sep=2pt,draw=green!70!blue]1}
-{}1{}-11{}-{}1-1{}1{}1{}111-{}1{}-1}
{
  \node[grp={blue}{1.9}{0.9}](n000) at (1.0,7.5) {};
  \node[grp={blue}{1.9}{0.9}](n002) at (7.0,7.5) {};
  \node[grp={blue}{1.9}{1.9}](n010) at (1.0,4.0) {};
  \node[grp={blue}{1.9}{1.9}](n012) at (7.0,4.0) {};
  \node[grp={blue}{1.9}{0.9}](n030) at (1.0,0.5) {};
  \node[grp={blue}{1.9}{0.9}](n032) at (7.0,0.5) {};
  \draw[blue] (n000.east) to [bend left=25] (n002.west)
    (n010.east) to [bend left=25] (n012.west)
    (n030.east) to [bend right=25] (n032.west)
    (n000.south) to [bend right=25] (n010.north)
    (n002.south) to [bend left=25] (n012.north)
    (n010.south) to [bend right=25] (n030.north)
    (n012.south) to [bend left=25] (n032.north);
  \node[grp={red}{0.8}{7.8}](n100) at (0.5,4.0) {};
  \node[grp={red}{0.8}{7.8}](n101) at (3.5,4.0) {};
  \draw[red] (n100.north) to [bend left=25] (n101.north)
    (n100.south) to [bend right=25] (n101.south);
  \node[grp={orange}{1.9}{1.9}](n200) at (5.0,6.0) {};
  \node[grp={orange}{1.9}{1.9}](n220) at (5.0,2.0) {};
  \draw[orange] (n200.west) to [bend right=25] (n220.west)
    (n200.east) to [bend left=25] (n220.east);
  \node[grp={teal}{1.8}{1.8}](n300) at (6.0,6.0) {};
  \node[grp={teal}{1.8}{1.8}](n320) at (6.0,2.0) {};
  \draw[teal] (n300.west) to [bend right=25] (n320.west)
    (n300.east) to [bend left=25] (n320.east);
}
\draw[<-] (Nv) - +(-1,1) node[lbl]{variable on its bar};
\draw[<-] (Nc) - +(-1,-1) node[lbl]{special cell};
\draw[<-] (Nl,120) - +(-1.1,3.5) node[lbl]{label for special cell};
\end{tikzpicture}

```

The corresponding Karnaugh map looks like this:



end the logic expression<sup>8</sup> is

$$z = \bar{x}_3 \bar{x}_1 + \bar{x}_2 \bar{x}_0 + x_3 x_2 x_1 + x_3 x_2 x_0.$$

You may notice that the zeros were omitted (replaced by {} in the list). Also, the cell 34 is special because `{[red,name=Nc, label={[name=N1,orange!90!black, label distance=1\kmunitlength]left:Special}, circle, inner sep=2pt, draw=green!70!blue]1}`. You can put almost anything inside a cell using curly brackets and you can customize the cell style using square brackets. The format is: `{[opt]string}` where `opt` is an optional set of styles (among other TikZ parameters) which will be passed as the last option of TikZ command `\node` and `string` will be written inside the cell by that command. To use this syntax, it is imperative that the very first character after the opening curly brackets (f) be the opening square brackets ([). Matching pairs of square brackets are allowed inside the optional sequence provided that they are protected inside a pair of curly brackets. In this case, the proper content of cell 34 is just the number 1 near the end, all the rest is the style applied to this single 1, therefore coded between square brackets. The style uses TikZ syntax in order to change colour, font size, add a label, add figure, add decoration and name it for future reference. In this case, two nodes are named Nc and N1 for future reference. Near the TikZ environment end, those names are used to place arrows pointing to the nodes with a description. The `\draw` command that draws those arrows cannot be placed inside the fifth argument of macro `\karnaughmap` because the fifth argument is typeset before the cell's contents (the fourth argument), therefore no name would be created at the time the fifth argument is typeset.

The variables identifiers (the third argument) can also be formatted individually using style, but note that the custom style will be applied to both the bar line and the node for the variable identifier. If a bar gets segmented, just like  $x_3$  bar, the named node will be the top most if the bar is vertical or the right most if the bar is horizontal. The  $x_3$  bar is different from the other bars because `[yellow!70!black,name=Nv,|-|,double=red, very thick, label={font=\tiny,green!50!black}above:var.}, text=blue!60!red]` changes its appearance. The node name Nv is also not available at the time the fifth argument is typeset. So, any command that makes use of it will need to be placed after the end of macro `\karnaughmap`.

The distance between bars on the left side was set to `1.2\kmunitlength` to prevent overlapping between  $x_3$  (the label) and  $x_4$  bar and  $x_4$  and  $x_5$  bar, but the distance between the bars on top was left unchanged. The distance between the map and the bars closest to it was set to `0.4\kmunitlength` to prevent overlapping between the bar tip ( $\Rightarrow$ ) and the map itself.

<sup>8</sup>This is not of any importance here, but I couldn't hold myself back. By the way, if you are curious, there are another two minimal solutions.

The indices can be computed by

$$32x_5 + 8x_4 + 2x_3 + 16x_2 + 4x_1 + 1x_0$$

which is a bit bizarre. The truth table values ought to be arranged according to this index order. This bizarreness is the price we pay to have the variables placed in positions which are more intuitive. See Section 8 for a java software that can help on this matter.

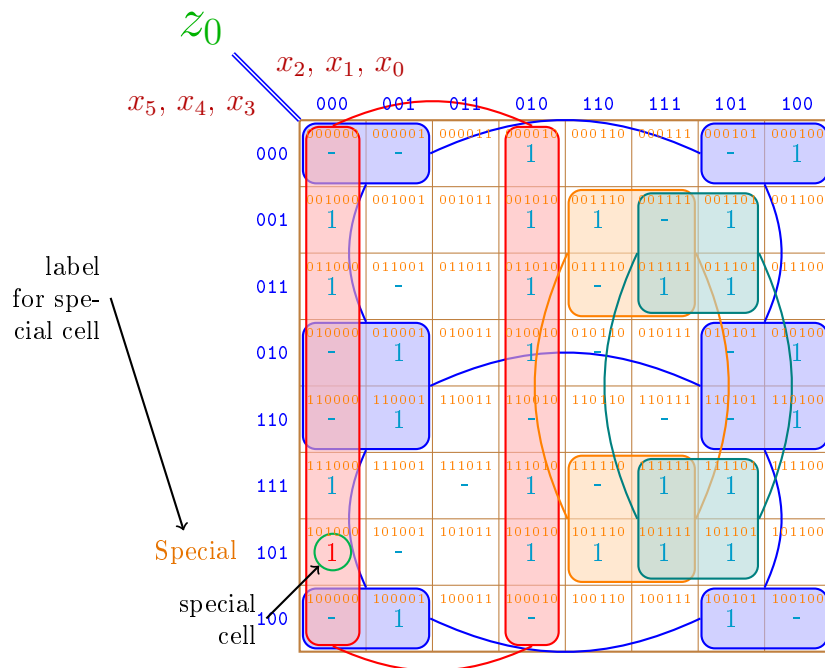
## 6 American style

Support for American styled map was included in version 1.3 and improved in version 1.4. Although the default style is what Karnaugh calls “simplified labels” [1], it is sufficient to enable the `American style` to obtain a map that resembles the maps found in many books on Digital Electronics and Digital System Design. You can use `\tikzset` to set this and other styles globally, for instance:

```
\tikzset{every karnaugh/.style={
  American style,
  kmindex/.style={orange,font=\tiny},
  enable indices,
  Gray index,
  kmindex pos={0.5}{0.8},
  kmsep line/.style={blue,double=white,semithick}}
}
```

enables the `American style` and, consequently, disables bars; sets the style for indices; enables index in each cell; sets the index to in Gray code; sets the index position and sets the style for the line that separates row from column variables lists.

The example of the last section, just deleting some useless code, renders as:



One possible problem is that, depending on the number of variables, one would need to reduce the labels on top of the map to an unreasonable size to prevent them from overlapping, or to enlarger the cell size to make them fit. The same is true for indices inside the cells.

If, instead of `Gray index`, you use `binary index` you would notice they are quite different. This is because binary indices reflect the position of the cell as it was taken from a row of a linear truth table. Since the algorithm that builds the map distributes the variables among the rows and columns of the map in a non-trivial way, the indices depend on the cell position and its relation to the variable input order. Gray coded indices are fix. The decimal indices can still be computed by

$$32x_5 + 8x_4 + 2x_3 + 16x_2 + 4x_1 + 1x_0.$$

If you are struggling with the variable position and the sequence in which variables and values should appear in order to result in the map you want, see Section 8 for a java software that can help.

## 7 Vertical mode

For an odd number of variables, the Karnaugh map is rectangular and macros `karnaughmap` and `karnaughmaptab` will typeset it twice as wide as it is high (not taking into account the bars or other features outside the grid). Like this single variable map:

$f(a)$   

$0$	$1$	$1$	$0$
-----	-----	-----	-----

```
\tikz[karnaugh,enable indices]{
  \karnaughmap{1}{f(a)}{{a}}{10}{}
```

This layout is good for presentations because the projection area is usually wider than higher. Paper sheets, on the other hand, are usually higher than wider, so for a big map you may need something like<sup>9</sup>:

$f(a)$   

$0$	$1$
$1$	$0$

```
\tikz[karnaugh,enable indices]{
  \karnaughmapvert{1}{f(a)}{{a}}{10}{}
```

This is called, for lack of a better name, vertical mode<sup>10</sup> and it is done by the `\karnaughmapvert` and `\karnaughmaptabvert` macros. Note that `\karnaughmapvert` and `\karnaughmaptabvert` macros arrange the variables in a different order. Compare the two square (four variables) maps below in the normal (on the left) and vertical mode (on the right) paying attention to the indices and variable identifiers.

$f(a, b, c, d)$   

				$b$			
				$d$			
$0$	$1$	$5$	$4$	$0$	$1$	$1$	$1$
$2$	$3$	$7$	$6$	$1$	$1$	$1$	$1$
$10$	$11$	$15$	$14$	$1$	$1$	$1$	$0$
$8$	$9$	$13$	$12$	$1$	$0$	$1$	$1$

Normal (horizontal) mode

$f(a, b, c, d)$   

				$a$			
				$c$			
$0$	$2$	$10$	$8$	$0$	$0$	$1$	$1$
$1$	$3$	$11$	$9$	$1$	$1$	$1$	$0$
$5$	$7$	$15$	$13$	$0$	$1$	$1$	$1$
$4$	$6$	$14$	$12$	$1$	$1$	$0$	$1$

Vertical mode

The indices are calculated in the same way, but their position inside the map is different because the variables positions are different. It is like one map is mirrored and then rotated 90° (mirrored horizontally and rotated clockwise or mirrored vertically and rotated anticlockwise.) Exactly like matrix transposition.

One interesting application of vertical mode is when you want to keep consistency in variable identifier position among maps with odd and even number of variables. For example, if you want the most significant variable  $a$  appearing on top of the maps you can use normal (horizontal) mode for maps of odd number of variables and vertical mode for even amounts, like this:

$f(a, b, c)$   

				$a$			
				$c$			
$0$	$1$	$5$	$4$	$0$	$1$	$1$	$1$
$2$	$3$	$7$	$6$	$1$	$1$	$1$	$1$

Normal (horizontal) mode

$g(a, b, c, d)$   

				$a$			
				$c$			
$0$	$2$	$10$	$8$	$0$	$0$	$1$	$1$
$1$	$3$	$11$	$9$	$1$	$1$	$1$	$0$
$5$	$7$	$15$	$13$	$0$	$1$	$1$	$1$
$4$	$6$	$14$	$12$	$1$	$1$	$0$	$1$

Vertical mode

<sup>9</sup>Or you can use landscape.

<sup>10</sup>Not to be confused with TeX vertical mode.

A more general approach is to use the java software described in Section 8 to create maps with arbitrary variables positioning. Suppose that you desire the most significant variable  $a$  to appear at the left side of a three variables map. You can do the opposite of what was done in the last example, but you will end up with a vertical map of three variables and maybe it is not what you want. Using the software described in Section 8 allows  $a$  to be placed at the left in a normal (horizontal) mode map. This will change the indices because it reorders the truth tablet such that  $a$  will no longer be the most significant variable, but without changing the logic function.

## 8 Java Quine McCluskey – JQM

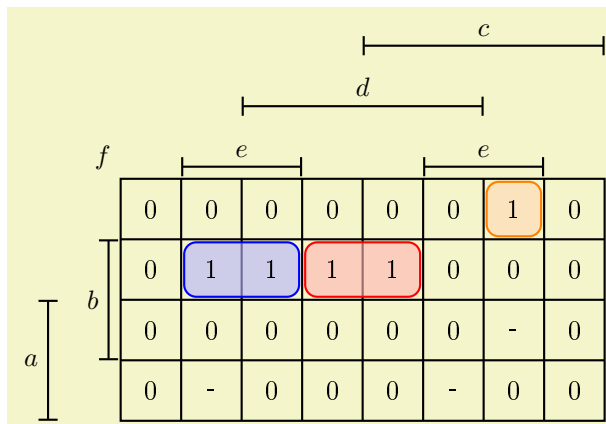
Java Quine McCluskey (JQM) implements the Quine McCluskey algorithm with Petrick’s Method (or the method of prime implicants) for minimization of Boolean functions. It is available on <https://sourceforge.net/projects/jqm-java-quine-mccluskey/>. Up to sixteen functions of sixteen variables can be minimized. A graphical interface is provided for entering and editing the truth table that can be saved and loaded.

One very useful feature of JQM is that you can reorder the variables on the map to suite your particular application instead of relying exclusively on the macro to scatter your variables around.

You can also input a map in text mode and obtain the TikZ code for creating the corresponding map, and the correspondence would be self-evident. For instance, a function  $f(a, b, c, d, e)$  can be written in a text file like:

```
.k
a,b,c,d,e,,f
0000 0010
0111 1000
0000 00-0
0-00 0-00
```

where - means don’t care and  $f$ , the output variable, is separated from the input variables by two commas. It would be rendered as:



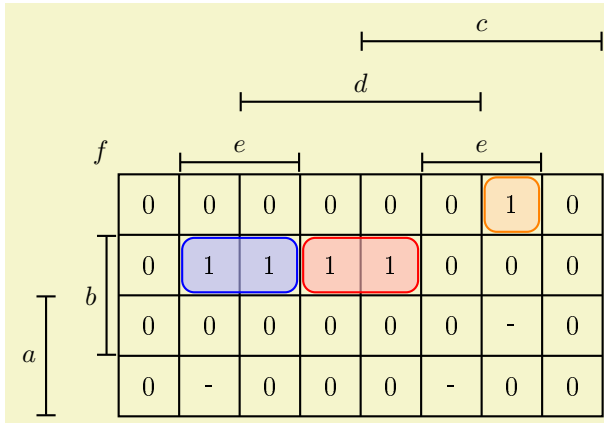
```
\tikzset{
  grp/.style n args={3}{
    draw=#1,fill=#1!30,
    minimum width=#2|kmunitlength,
    minimum height=#3|kmunitlength,
    rounded corners=0.2|kmunitlength,
    fill opacity=0.6,
    rectangle}
}

\begin{tikzpicture}[karnaugh,thick]
\karnaughmap[5]{f}{c}{d}{a}{b}{e}{%
{000100110-00000001000010000-0-00}%
{
  \node[grp={blue}{1.9}{0.9}] at (2.0,2.5) {};
  \node[grp={red}{1.9}{0.9}] at (4.0,2.5) {};
  \node[grp={orange}{0.9}{0.9}] at (6.5,3.5) {};
}
}
\end{tikzpicture}
```

Note that the original text file represents the obtained Karnaugh map very faithfully, but the command that creates the map above, in particular parameters #3 and #4, has no evident relation with the original text or the generated map. This, again, is due to the recursive algorithm that scrambles the variables.

Observe the sequence order of variables  $a, b, c, d$  and  $e$  in the text file and in the map: there is a clear correspondence. However, the sequence order has to be modified in parameter #3 to compensate for scrambling produced by the algorithm and this make the manual encoding of parameter #4 complicated. For a small map, meaning a small number of variables, it would not be too difficult to manually sort the parameters in order to obtain the desire result, but it would be simply too problematic for a bigger map. That’s where JQM comes in. You can manipulate the map assigning the position of the variables in any order and the software automatically sorts the truth table accordingly. The order in which the variables will appear in the final expression is also independent on the order they appear in the map.

If, however, the export option ‘Table mode (new algorithm)’ is activate in JQM (it is new in version 1.3.4) the result would be very similar to the input file. This is because the new macro `\karnaughmaptab` is designed to keep the parity between the Karnaugh map and macro parameters, see below:



```

\tikzset{
  grp/.style n args={3}{
    draw=#1,fill=#1!30,
    minimum width=#2|kmunitlength,
    minimum height=#3|kmunitlength,
    rounded corners=0.2|kmunitlength,
    fill opacity=0.6,
    rectangle}
}

\begin{tikzpicture}[karnaugh,thick]
\karnaughmaptab{5}{\f$}{\a$}{\b$}{\c$}{\d$}{\e$}}%
{00000100111100000000-00-000-00}%
{
  \node[grp={blue}{1.9}{0.9}] at (2.0,2.5) {};
  \node[grp={red}{1.9}{0.9}] at (4.0,2.5) {};
  \node[grp={orange}{0.9}{0.9}] at (6.5,3.5) {};
}
\end{tikzpicture}

```

Another popular way of inputting the truth table is to provide a list of indices for which the output is one or *don't care*. Alternatively, it can be zero and don't care. Consider the example shown in Table XIV of reference [3]. Given the binary variables  $x_4, x_3, x_2$  and  $x_1$  where  $x_4$  is the most significant, it is desired to input the decimal equivalents of the binary numbers formed by the entries (called *indices* herein) that lead the output to one (or zero) and some combinations of input conditions for which the output is not specified (don't care or *d*-entries). In this example, the decimal inputs that lead to ones are 5, 6 and 13 and the *d*-entries are 9, 14. In the notation used by McCluskey [3]:

$$T = \sum (5, 6, 13) + d(9, 14) \text{ where 9 and 14 are the } d\text{-terms.}$$

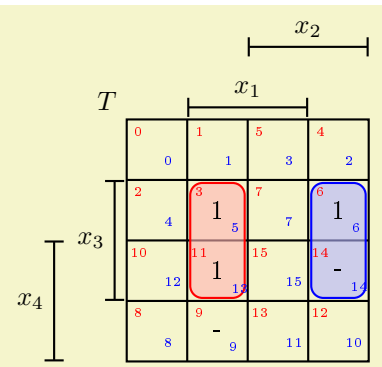
The input file should contain:

```

.L
x_4,x_3,x_2,x_1,,T
s, 5, 6, 13, d, 9, 14

```

where the first line has the command .L (lower case L) to indicate *list mode*. In the second line there is a list of input variables (sequence order is ultimately important here) and the output variable  $T$ , again, separated from the input variables by two commas. The third line has a list of indices that lead the output to one starting with the letter s (should be z for a list of zeros), then the letter d which indicates the beginning of the *d*-entries, and finally the *d*-entries. From this text file we obtain:



```

\tikzset{
  grp/.style n args={3}{
    draw=#1,fill=#1!30,
    minimum width=#2|kmunitlength,
    minimum height=#3|kmunitlength,
    rounded corners=0.2|kmunitlength,
    fill opacity=0.6,
    rectangle}
}

\begin{tikzpicture}[karnaugh,x=1|kmunitlength,
y=1|kmunitlength,thick,
enable indices,
kncell/.style={
  minimum height=1.5em,
  label={font=|tiny,blue,
  label distance=-0.3|kmunitlength
  below right:|kmdectoKGdec|kmindecxcounter}}]
\karnaughmap{4}{T$}{\x_4$}{\x_2$}{\x_3$}{\x_1$}}%
{0-0-01-0-01-0-0-01-0-0-0}%
{
  \node[grp={blue}{0.9}{1.9}] at (3.5,2.0) {};
  \node[grp={red}{0.9}{1.9}] at (1.5,2.0) {};
}
\end{tikzpicture}

```

where the indices were enabled and a secondary index was inserted in every cell. This secondary index, shown in blue, uses the macro `\kmdectoKGdec` to convert the decimal index given by `\kmindecxcounter` to a pseudo

index that is equal to the decimal equivalents of the binary numbers formed by the entries  $(x_4, x_3, x_2, x_1)$ . This is not the index used to assemble the map which is shown in red, is equal to `\kmindexcounter` and is the decimal equivalent to  $(x_4, x_2, x_3, x_1)$ . JQM rearranged parameters #3 and #4 such that, as in the previous example, the map is typeset accordingly to the specification given in the input text file. To force the macro `\karnaughmap` to behave as expected, JQM swapped variable  $x_2$  and  $x_3$  and resequenced parameter #4. This is done automatically and the user does not have to worry with the details. As expected, the result shows  $x_4$  and  $x_3$  assigned to the rows (at the left-hand side of the map) and  $x_2$  and  $x_1$  assigned to the columns (on the top of the map).

## 8.1 JQM options to export to L<sup>A</sup>T<sub>E</sub>X file

When exporting a L<sup>A</sup>T<sub>E</sub>X file from JQM, bear in mind that if a minimized truth table is exported, the corresponding map, or maps for multiple outputs or multiple solutions, will have the solution highlighted. Conversely, non-minimized truth tables will not present the solution although they can be exported.

There are several options that impact on the ".tex" file produced. First of all, you can either produce a complete document or just an insert to be included in a master document. Next you see options regarding Boolean expressions, truth table and ladder diagram (PLC program). Interested readers can find the description in JQM ReadMe file.

Karnaugh maps options start given the possibility of creating maps with both zeros and ones. The default behaviour is to replace zeros (in a ones map, sum of products, disjunctive normal form, or OR of ANDs) or ones (in a zeros map, product of sums, conjunctive normal form or AND of ORs) by empty cells. The intention is to keep the map as clean as possible. By checking the box, both zeros and ones will appear on the map.

Next option gives the possibility of generating the Boolean expression. Moreover, this expression is colour coded: each implicant is generated in the corresponding colour of the highlight on the map.

Then you have the choice between truth table mode (which uses `\karnaughmap`) and table formatted Karnaugh map (which uses `\karnaughmaptab`). The preselect option favours `\karnaughmaptab`. Although the macro used is different, the generated map is the same because JQM arranges the data sent to the macro accordingly. Thus, if you like to see in your ".tex" file how the map will look like, use the table mode. If you prefer to see the truth table uncheck the box.

Vertical mode, see Section 7, is supported. In this mode, the map will look like it has been transposed. The variables associated with rows will be associated with columns vice-versa. A rectangular map (with an odd number of variables) will look twice as high as it is wide in this mode and that is why it is called *vertical mode*. For a square map (with an even number of variables), the same effect can be obtained reordering the input variables<sup>11</sup>.

You can generate tests (exams) using package `examdesign`. This package can be used to create different tests versions of the exam preventing cheating. The support included in JQM for `examdesign` allows the generation of answer sheet with solution and exam (obviously, without solution).

Finally, we have a button that opens a new window for selecting the order in which the variables will appear on the map. There are three preestablished orders, but the user is free to move the variables around assorting them in any order. Note that the input variable order, which can be changed in the main window, affects the variables position in both Boolean expression and Karnaugh map. Thus, it is always a good idea to verify the desired order before exporting.

Users are encouraged to experiment with those options and see which ones fit some particular need.

## 8.2 JQM main features

- Up to 16 input variables.
- Up to 16 functions (output variables).
- Petrick's Method used to find solutions covered by non-essential prime implicant.
- Comfortable graphical interface allows variable renaming and column reorder.
- Truth table can be saved in CSV file for external editing or reuse. Then it can be loaded again. Also, one can generate the truth table using other software and "import" (open) for edition and minimization.

---

<sup>11</sup>There is a tiny difference that appears when implicants are segmented in several parts particularly in big maps. In this case, some lines that connect those parts are vertical instead of horizontal.



- Besides using the graphical interface, truth table can be written in a text file then load in the software. Several formats are available including: list decimal representing implicants and don't care, Karnaugh map and CSV with wildcards.
- Results can be expressed in several formats like: conventional Boolean expression,  $\LaTeX$ , Structured Text (ST) and Ladder Diagram (LD).
- Results can be exported to HTML file and open in an internet browser.
- Because this software aims PLC programming, solutions are independent (non-simultaneous). It would not be too difficult to modify the algorithm to support simultaneous solution.
- Generates Karnaugh maps in HTML and  $\LaTeX$ .
- Not only solves the problem, but also shows how the solution was obtained.
- To use the software just download the zip file, unzip it and double click on "JQM-QuineMcCluskey.jar". Please see ReadMe.txt, or LeiaMe.txt if you prefer Portuguese, for more details and examples.

## 9 Final remarks

To effectively typeset good looking Karnaugh maps you may need to know more about *TikZ* and this material is beyond the scope of this text, but you can refer to [2].

In case you find a bug, or if you have comments or suggestions, please send me an e-mail.

The maximum size map I could produce was a Karnaugh map with twelve variables; with bigger maps I only exceeded  $\TeX$ 's main memory. This is due to the huge amount of *TikZ* nodes necessary to create the map. Quite likely you will exceed  $\TeX$ 's capacity with even smaller maps if they occur in large documents.

## References

- [1] **Karnaugh**, Maurice. The map method for synthesis of combinational logic circuits, *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics* 72(5), 593–599, 1953.
- [2] **Tantau**, Till. *The TikZ and PGF Packages* : Manual for version 3.1.9a-34-ga0d9cada, <https://github.com/pgf-tikz/pgf>, 2021.
- [3] **McCluskey**, Jr., Edward Joseph. Minimization of Boolean functions, *The Bell System Technical Journal* 35(6), 1417–1444, 1956.