An Application for Solving Truth Functions

Pavel Pokorný[1] and Daniel Ševčík[2]

[1] Department of Computer and Communication Systems
Tomas Bata University in Zlín, Faculty of Applied Informatics
Nad Stráněmi 4511, 760 05 Zlín, Czech Republic
pokorny@utb.cz
[2] Department of Automation and Control Engineering
Tomas Bata University in Zlín, Faculty of Applied Informatics
Nad Stráněmi 4511, 760 05 Zlín, Czech Republic
d_sevcik@utb.cz

Abstract. Nowadays, many technical devices are controlled by logic circuits. These circuits evaluate the situation based on the proposed truth functions, which they then use to perform one of the defined actions. In line with the increasing complexity of individual technical devices, the logic circuits and truth functions are also increasingly complex. For this reason, it makes sense to minimise these functions since they can thereby achieve a simpler technological process and the higher reliability of whole logic systems. This paper describes the Karnaugh Studio application, which resolves the minimisation of these truth functions - and was developed at our faculty (FAI, TBU in Zlín). The minimisation is performed using the Karnaugh Map Method - with support for up to eight variables on input. The application is based on the C ++ programming language and the Dear ImGui and Magick ++ libraries. Its functionality has been verified on a number of examples. This proved its applicability and ability to be used in the solution of logic circuits in industrial practice.

Keywords: Minimisation, Truth Function, Boolean Function

1 Introduction

Human beings usually perform arithmetic operations using the decimal number system, - but, by comparison, a digital machine, (for example a computer), is inherently binary in nature and its numerical calculations are executed using a binary number system. [1]

In a computer, the input is given with the help of switches. This is then converted into electronic signals, which always have two distinct discrete levels or values – a high level and low level. As long as the signal is within a pre-specified range of high and low, the actual value of the signal is not that important. Such signals are called digital signals and the circuit within the device is called a digital circuit. These concepts are examples of a digital system and they can be applied in computers, telephony, radar navigation, data-processing, and many other applications. [2] [3]

A digital logic system may well have a numerical computation capability as well as its inherent logical capability and consequently, it must be able to implement the four basic arithmetic processes of addition, subtraction, multiplication and division. [1]

These digital systems are usually made up of large assemblies of logic gates, which are often printed on integrated circuits, and controlled by truth (logic) functions. Truth functions are functions that accept truth values as input and produce one truth value as output.

Boolean Algebra is a mathematical system that forms the base for these operations. Boolean Algebra uses three basic logic operations – NOT, AND, OR. The NOT operator takes a single output and generates one output – it inverts the input value. The AND operator of two variables as input generates 1 as output if both of the variables are 1. The OR of two variables as input generates 1 as output if either - or both, of the variables are 1. In addition, one other function is required for arithmetic-related operations – XOR (the exclusive OR). The XOR of two variables as input generates 1 as output if either of them - but not both, is 1. [1]

A Boolean function consists of a number of Boolean variables joined by the Boolean connectives AND and OR. These functions can be represented by a Truth Table. The true table contains a row for every combination of the input logic values and prescribes the output value of the function for each of these combinations.

One of the main objectives of engineers when designing discrete gates is to keep the number of gates to a minimum when the Boolean function is implemented. A Karnaugh Map represents an easy method of simplification – with the help of a number of simple rules, it allows one to reduce the Boolean function to its minimal form. Karnaugh maps correspond to the Truth Table - as Figure 1 illustrates. The values inside the squares are copied from the output column of the Truth Table – there is one square in the map for every row in the Truth Table.

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

AB \ CD	00	01	11	10
00	1	1	1	0
01	0	1	0	0
11	0	1	0	0
10	1	0	1	1

Fig. 1. The Truth Table (left) and their Karnaugh Map (right) for four input variables

This paper describes the application that resolves the minimisation of the described truth functions, using the Karnaugh Map method. Chapter 2 describes minimisation methods of truth functions. Chapters 3 contains the developer application information – selected the appropriate programming language and the libraries used. More details like application possibilities, the implementation of the minimisation algorithm and the application user interface are mentioned in Chapter 4.

2 Truth Function Minimisation Using Karnaugh Maps

Before we describe basic rules for truth function minimisation using Karnaugh maps, it would be good to mention the advantages and disadvantages of these maps. The advantages include simplification for a small amount of input variables and always results in minimum expression if performed properly, prevention of the need to remember each and every Boolean algebraic theorem, and the involvement of fewer steps than the algebraic minimisation technique to arrive at a simplified expression. Among its disadvantages however, are the complicated solution process as the number of input variables increases and the minimum logical expression may - or may not, be unique depending on the choices made while forming the groups. [4]

The number of cells in the Karnaugh Map is expressed as two raised to the power of the number of input variables, i.e. with two inputs, we have four cells, with four input-value sixteen cells are necessary, and so on. [5]

The main key to using Karnaugh Maps to find truth function minimisation identify “groups” of 1's (and no 0's) or groups of 0's (and no 1's) on these maps. A valid group must be a “power of 2” size - (meaning that only groups of 1, 2, 4, 8, or 16 are allowed), and it must be a square or rectangle - but not a dogleg, diagonal or other irregular shape. According to the specified group, each '1' or each '0' must participate in at least one group, and each '1' or each '0' must be in the largest possible group. The requirement that all 1's (or 0's) are grouped in the largest possible group may mean that some 1's (or 0's) are part of several groups. [6]

In practice, loops are drawn on a K-map to encircle the 1's (or 0's) in a given group. Once all 1's (or 0's) in a map have been grouped in the largest possible loops, the grouping process is complete and a logic equation can be read directly from the Karnaugh Map. If the procedure is performed correctly, a minimal logic equation is guaranteed.

As mentioned above, the minimisation algorithm can have two forms. The Sum-of - Products form if we select group 1's (and no 0's), and the Product-of-Sum form if we select group 0's (and no 1's). [7]

Using the Sum-of-Products form, the variables are operated by AND (product) to form a product term and all these product terms are ORed (summed or added together) to get the final function. The Sum-of-Products form is also called the Disjunctive Normal Form and is most suitable one to use in FGPA (Field Programmable Gate Arrays).

The Product-of-Sum means that all the variables are ORed, i.e. written as sums to form sum terms. All these sum terms are ANDed (multiplied) together to get the product-of-sum form. This form is exactly the opposite to the Sum-of-Products form and is also called the Conjunctive Normal Form.

3 Developer Tools Used

The C++ programming language was used to develop an application that resolves truth function minimisation. This language was chosen based on personal experience and a number of benefits that it offers. C++ is an open ISO-standardised compiled language. It compiles directly into a machine's native code, allowing it to be one of the fastest languages in the world, if optimised. C++ is a language that is directly built on C language and is compatible with almost all C code. C++ can use C libraries with few to no modifications of the libraries' code. C++ has a wide range of compilers that run on many different platforms that support it. [8]

C++ is a language that expects the programmer to know what they are doing, but also allows the programmer to have incredible amounts of control as a result. As for the latest C++ standard, C++ supports both manifest and inferred typing, allowing flexibility and a means of avoiding verbosity where desired. C++ allows one-type conversions to be checked either at run-time or at compile-time – again, offering another degree of flexibility. Most C++ type checking is however, static. C++ offers remarkable support for procedural, generic, and object-oriented programming paradigms, with many other paradigms being possible as well. [9]

Applications developed in the C++ language can use thousands of additional libraries that will run on many platforms with few to no changes. One of the most used is the Standard Template Library (STL) - which we used in our application. STL is a set of C++ template classes that provide common programming data structures and functions like lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalised library and so, its components are parameterised. STL has four components: [10]

- Algorithms – a collection of functions especially designed to be used on ranges of elements (sorting, searching, useful array algorithms …).
- Containers – these store objects and data (sequence containers, associative containers, container adaptors and unordered associative containers).
- Functions – classes that overload the function call operator. They allow the working of the associated function to be customised with the help of parameters to be passed.
- Iterators – are used to point at the memory addresses of STL containers. Iterators are primarily used in number or character sequences and reduce the complexity and execution time of the programme.

Another library we used was Dear ImGui [11]. Dear ImGui is a bloat-free graphical user interface library for the C++ language. It is fast, portable, renderer agnostic and self-contained - (no external dependencies). ImGui is designed to enable fast iteration and empower programmers to create content creation tools and visualisation/ debugging tools (as opposed to UI for the average end-user). It favours simplicity and productivity toward this end, and thus lacks certain features normally found in more high-level libraries. [12] We used this library in order to make the user interface in our application.

Magick ++ was the last library we used. It represents a free object-oriented C++ API and provides integrated support for the Standard Template Library. It’s primarily use is for creating, editing, composing and converting bitmap images. It supports a variety of

graphic formats (over 200), including the most commonly used - for reading and writing. [13] This library was used to work with bitmap images in our application.

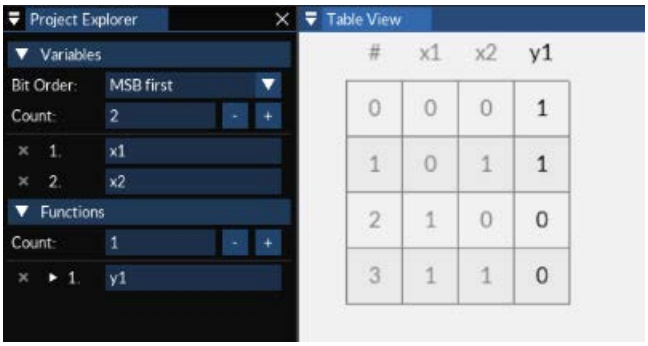

Fig. 2. A manually set truth table in the Karnaugh Studio

4 An Application

The truth function minimisation application is called the Karnaugh Studio. Its purpose is to perform both manual and fully automatic minimisation on up to 32 functions in a single project, with a maximum of eight input variables. The user interface is designed as fully graphical, simple, intuitive, and fully customisable. One can set layout, colour scheme, captions, and other properties for the truth table and generated Karnaugh maps. In order to process project and configuration files, Karnaugh Studio uses the JSON file format. The application is free for use and can be download from the internet [14].

The user can set input with three methods. They can fill the truth table by hand (Fig. 2); import it from the CSV file; or define the functions using the Sum-of-Products sums form or the Product-of-Sum form. When the table is set, one or more Karnaugh maps are automatically displayed in dependency of the input parameters. After that, the user can make groups of 1's or 0's manually using an automatic algorithm - described in Chapter 4.1. The same algorithm minimisation displays the results.

In addition, the counted results can also be exported, validated and saved in a wide variety of formats - including many programming and mark-up languages (C/C++/C#/Java, Python, LaTeX, etc.). The truth table and Karnaugh maps can also be exported as either rasters (PNG, JPEG, TIFF, BMP) or vectors (SVG) image.

4.1 The Minimisation Algorithm

The Minimisation algorithm is composed of four steps: minimisation group creation, additional minimisation, generating expressions, and expression validation.

During the minimisation group creation process, it is very important to select the correct Karnaugh map creation method for functions with more than four input variables.

Complications can be expected since the application supports up to eight input variables. Due to this - the most practical option was implemented. This divides data into

several completely independent maps. The number of maps depends on the number of input variables. A single Karnaugh map is used for up to four input variables. The application generates two maps for five input variables; four maps for six variables (Fig. 3); eight maps for seven variables; and sixteen maps for eight input variables. This division makes possible the application of the same algorithm to each map separately - regardless of the total number of input variables and other maps.

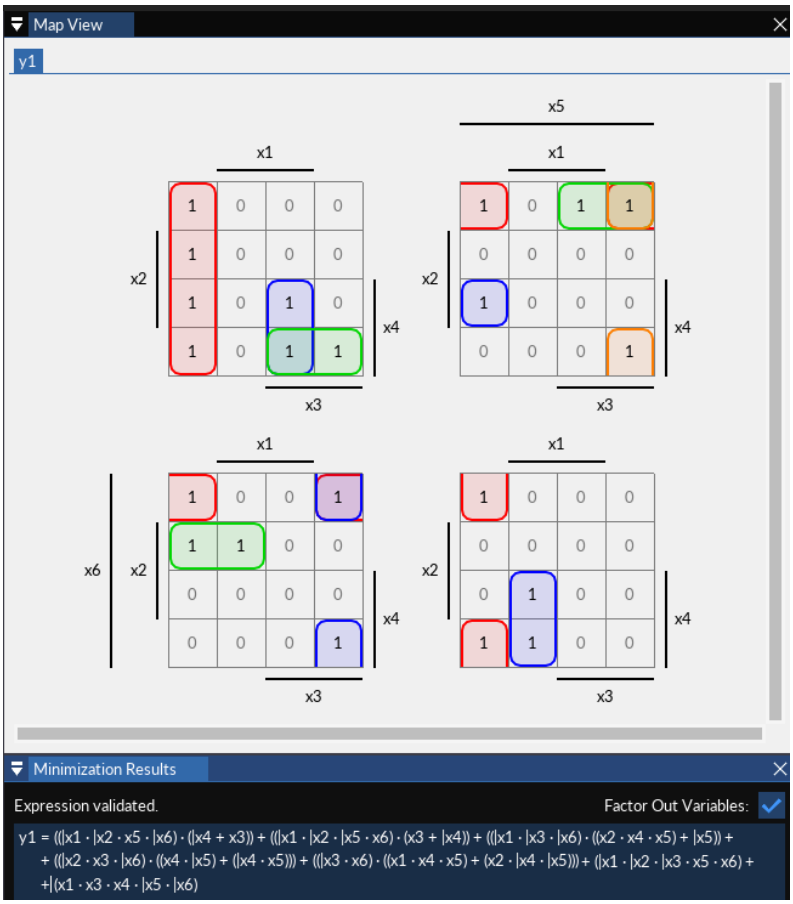

Fig. 3. An example of Karnaugh maps for 6 input variables and minimisation results

The algorithm starts by creating all valid minimisation group combinations for each Karnaugh map separately. A basic group is created for each element in which the logical one is located - from which, all possible combinations are gradually obtained by recursively extending them in all directions. When the groups are created, only an appropriate number of them are selected, based on several criteria. First of all, is necessary to remove duplicated groups in order to avoid unnecessary processing. After obtaining

all unique groups, it is necessary to combine identical groups across multiple maps into one element and mark them on the maps on which they are located.

The selection of minimisation groups from all those on offer is the most important step of the whole process. Deciding which group to use - or not to use, is based on two criteria. The main criterion represents the addition of elements on all maps where the group can be located. This addition means that the number of all fields not yet included containing a truth value of one, would be newly included if used. The second criterion is activated only when two or more groups with the same addition are found. Here, it is necessary to select one of them on the basis of its size, multiplied by the number of maps on which it is located. Then, all connected and pre-sorted loops are separated for each map on which they are located. This step is implemented for ease of manipulation in the case of manual minimisation.

When all the necessary minimisation loops have been created, it is then necessary to create a functional form - in order to later be able to apply the basic rules of Boolean Algebra. The first step is to apply the basic rules of Boolean Algebra. It is not necessary to include more complicated rules in the form of De Morgan's Laws since the aim is to improve the previously obtained results of the minimised Karnaugh map method. It is still necessary to take into account that one can carry out the minimisation manually - and thus, create certain backlogs. Primarily, all duplicate elements need to be removed. This is followed by the removal of all of the variables in the list of sub-expressions; in both normal and negated forms – thus applying the Third Exclusion Law. The most important feature of the whole algorithm in this phase is the ability to set variables for further recursive minimisation - thereby further minimising the functional form.

The penultimate step of the whole minimisation process is to generate a textual depiction of its resulting expression. This is based on the user-selected output format, given by a set of text strings which define the shape of the elementary constructions in the forms of conjunction, disjunction, equality - and others.

The expression validation represents the final part of the minimisation algorithm. This phase checks if the generated expression is complete and corresponds to a specified function by its truth values, which should be reliably replaced. Validation´s greatest importance lies in the ability to notify the user of an incomplete manual minimisation result. However, this phase is also important for the development and subsequent testing of the minimisation algorithm.

Validation occurs every time the output expression is built -, even when minimisation results are exported to a file; but, it is possible to deactivate this validation in order to save time. It is a relatively time-consuming process whose activity consists in the gradual reconstruction of truth values from the created expression and their comparison with the original function values.

4.2 The Application Interface

When a user starts Karnaugh Studio, the welcome screen is rendered. Its purpose is to speed up access to the most common operations that a user normally performs after start-up. The most useful feature is to be able to quickly load one´s recently edited projects. Their list is located on the right-hand side of the window, and consists of up to five items. Each of them in the form of a button bearing the name and complete path

to the project file. Clicking on this will immediately load the selected project. Other window elements include buttons to load an existing project from a file, or to create a new project, or to restore a previous session that puts the programme in the state that the user left it in at the last exit.

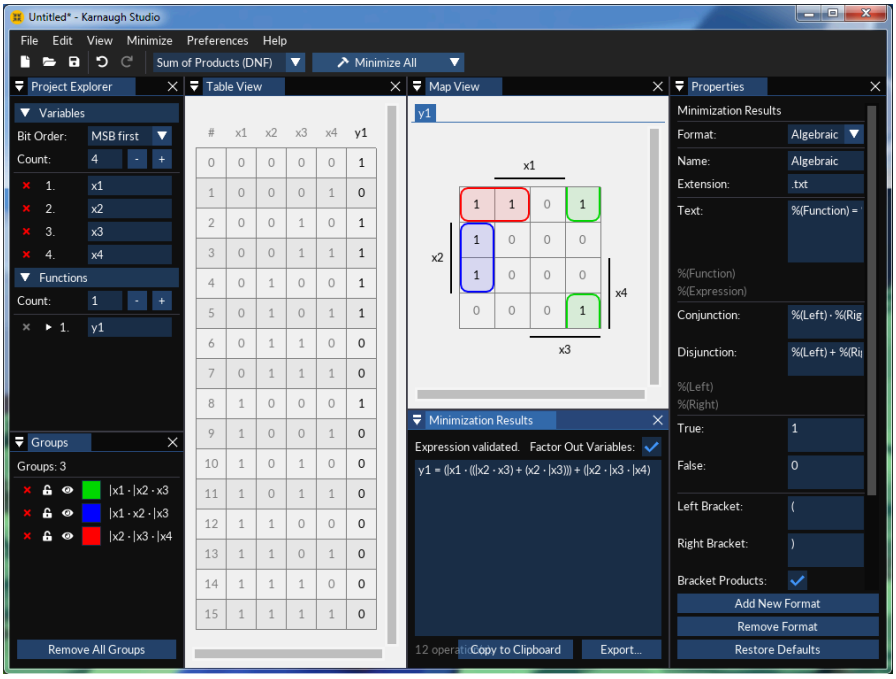

Fig. 4. The Karnaugh Studio user interface

The main application window is displayed when the user creates a new project or opens an existing project. This is shown in Fig. 4.

Above the placed main text menu is an integral part of almost every desktop application. Using it, the user can perform a variety of activities, whether working with the project or user interface windows, performing auto minimisation - or anything else. Most of these actions are assigned to hotkeys that can be used to invoke the action without requiring the user to enter the main menu. The menu is divided into six clearly defined submenus: File, Edit, View, Minimise, Preferences and Help.

The toolbar is located directly below the main menu. It consists of a set of icons and other elements that provide quick access to selected Main Menu actions. First of all, there are three project management icons. These are used to create a new project, load one from a file - or save an existing one. Undo and Redo edit command icons are located on the right. The following command represents a choice between the Sum-of-Products and Product-of-Sum forms minimisation solution. The last element is a button that performs automatic minimisation. This can be done - either for all functions at the same

time, or only for an active one. It is possible to switch between these modes by the last arrow-shaped button.

The Project Explorer window - (Fig. 2, left), is located on the left, and is one of the most important elements of the user interface. The only alternative for setting these parameters is to import the contents of the truth table from any CSV file. Here, it is possible to set up the most basic project properties – variables - (input values settings), and functions - (number of outputs, their names and limits).

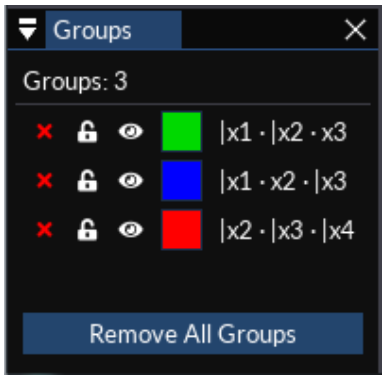

Fig. 5. The Groups window

The Groups window can be found under Project Explorer - which offers two different display modes. When a none group is created it describes basic instructions about how to create the first group. But this window primarily provides an overview of all existing minimisation groups in Karnaugh maps (Fig. 5). The current number of groups is visible at the top of the window. At the bottom, users can find a button to delete all groups at once. The central part of the window contains the list of created groups. Each group has its own name, based on input values, and has three properties. These are deleted, locked or invisible.

The Table View window (Fig. 2, right) is located to the right of Project Explorer. It provides a visual design of the project truth table. It represents an intuitive way for manually setting truth functions. This can be done by clicking the left-mouse button; this changes the output value in the cycle zero-one-indeterminate state – or, by right clicking the mouse button and selecting the required value from the displayed list.

The Map View window, (Fig. 3 above), is in the centre of the application window. The Karnaugh map is displayed here - and automatically updated based on output values, set in the Table View window. Additionally, the manual design and modifications of minimisation groups is only possible in this window. Manual design is performed in a simple way. A group can be created by double-clicking on any field, or by selecting multiple elements at the same time. However, it must always be true that at least one of the cells be - (the Sum-of-Products form), or if false, at least one of the (the Product-of-Sum form) cells. If it is not possible to create a group exactly matching the selection, a group is created with the most ideal coverage of the map elements inside the selection.

When the selection is complete, it is saved to the top of the active group list and is automatically assigned one of the predefined colour sets.

Using the Minimisation Results window, (Fig. 3, below) - Under Map View, the user can find the results of the minimisation algorithm in the selected output format. The result is displayed using a multiline text field, set to read-only. In the upper part of this window, users can observe the validation status of the generated expression. If incomplete or incorrect for any other reason, the user is notified by a red warning text. Otherwise, the text confirms the correct result.

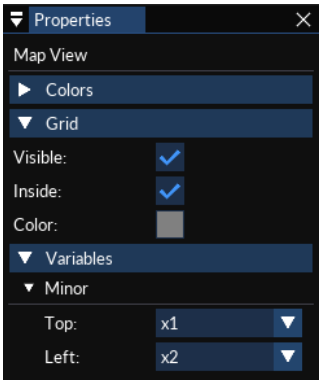

Fig. 5. The Properties window with settings for the Map View window

The last window is called Properties, and is positioned on the right-hand side of the application's interface. Using this, the user can set various settings for other windows – for example, visibilities, colours, headers, labels, grids, etc. Its content changes depend on the last active window of the three supported - Table View, Map View and Minimisation Results. Changes are immediately reflected in the active windows. The Properties window appearance for the Map View window is shown in Fig.5.

5 Conclusion

This paper describes the design and implementation of the Karnaugh Studio desktop application, which performs the minimisation of truth functions. The practical use of this application is in the field of design of truth functions and circuits, the optimisation of control systems - and last but not least, the academic sphere in support of teaching minimisation logic functions.

The Karnaugh map method is used as the algorithm for minimisation and to create application supports up to eight variables on input. The application also offers the possibility to perform the entire minimisation process automatically, or manually. In addition, Karnaugh Studio can export a number of graphic and text file formats, including programming languages commonly used in the developer company industry.

The aim is to further expand Karnaugh Studio with new features in the future. To add the ability to process an input function in any text format or to perform automatic

minimisation in real-time, while making changes inside the truth table. Another improvement may be to increase the visual quality of the graphical representation of the result of the minimisation process and to increase the possibilities of its graphic export. These visions would allow us a wider implementation of the application in practice.

References

1. Holdsworth, B., Woods, C.: Digital logic design. 4th edn. Newnes (2002). ISBN 978-0-750645-82-9.
2. Saha, A., Manna, N.: Digital principles and logic design. Laxmi Publications (2007). ISBN 978-1-934015-03-2.
3. Rafiquzzaman, M.: Fundamentals of Digital Logic and Microcomputer Design. 5th edn. Wiley-Interscience (2005). ISBN 978-0-471727-84-2.
4. The Karnaugh Map Boolean Algebraic Simplification Technique, https://www.allaboutcircuits.com/technical-articles/karnaugh-map-boolean-algebraic-simplification-technique/, last accessed 2020/01/18.
5. Prasad, V.C.: Generalized Karnaugh Map method for Boolean functions of many variables. IETE Journal of Education, 58(1), 11-19 (2017).
6. Learn Digilentinc – Logic Minimization, https://learn.digilentinc.com/Documents/319, last accessed 2020/01/18.
7. Boolean Functions (SOP, POS forms), https://www.electronicshub.org/boolean-logic-sop-form-pos-form/, last accessed 2020/01/18.
8. Cplusplus.com – The C++ Resources Network, http://www.cplusplus.com/, last accessed 2020/01/18.
9. Lippman, S. B., Lajoie, J., Moo B.E.: C++ Primer. 5th edn. Addison-Wesley Professional (2012). ISBN 978-0-321714-11-4.
10. The C++ Standard Template library, https://www.geeksforgeeks.org/the-c-standard-template-library-stl/, last accessed 2020/01/18.
11. GitHub - ocornut/imgui: Dear ImGui: Bloat-free Immediate Mode Graphical User interface for C++ with minimal dependencies, https://github.com/ocornut/imgui, last accessed 2020/01/18.
12. Dear imgui, https://skia.googlesource.com/external/github.com/ocornut/imgui/+/v1.51/README.md, last accessed 2020/01/18.
13. Magick++ API, https://imagemagick.org/Magick++/, last accessed 2020/01/18.
14. Karnaugh Studio, https://sevcikdaniel.github.io/karnaugh-studio/, last accessed 2020/01/18.