International Journal of Recent Engineering Research and Development (IJRERD)
ISSN: 2455-8761
www.ijrerd.com || Volume 03 – Issue 08 || August 2018 || PP. 63-70

Development of an Interactive Karnaugh Mapping Tool for Improving Digital Logic Education

Marcus Lloyde George[1], Monique Sampson[2]
University of the West Indies, Department of Electrical and Computer Engineering
St. Augustine, Trinidad and Tobago

Abstract: Several techniques can be used in Combinational Logic Design. One popular approach involves the use of Karnaugh Maps which can be used to derive logic equations for a given digital specification. The topic of Karnaugh Maps is normally introduced to students at the University of the West Indies, St. Augustine via lectures, however, there may be great merit in the use of technology in teaching topics such as Karnaugh Maps to students. In engineering education, it is important for students to understand this technique in order to design optimized digital circuits. This paper presents the development of a PC-based Karnaugh Mapping Tool that can be used in the teaching of combinational logic design to engineering undergraduates. The ultimate goal of the tool is to improve student performance at the University of the West Indies, St. Augustine (UWI) by providing students with a highly interactive tool. In addition, students can use the tool in their study time and learn at rate which is most suitable for them.

Keywords: Karnaugh maps, Karnaugh Mapping Tools, Digital Electronics, Digital Design, Combinational Logic, PC-based learning

1. Introduction

The Karnaugh Mapping [13] technique itself is a calculation usually done using pen and paper. It involves following a set of rules for circling the largest groups of ones possible until all the ones present for that given Karnaugh Map [13] problem is circled. From the circled group of ones the Sum of Products (SOP) [13] expression is deduced. This SOP expression describes the newminimized circuitry. The Karnaugh Mapping tool described in this paper will perform this technique for 2, 3 and 4 variable Karnaugh Maps. It will provide to the user the following:

- A step by step solution demonstration,
- The final solution,
- Options to view the implicants, prime implicants and essential prime implicants,
- A feature that identifies and solves the Static 1 and Static 0 hazards, and
- A feature that displays the minimum cover.

In order to successfully meet the above listed functions and features, the Quine McCluskey (QM) [9], [10] method incorporating Petrick's method [11] was selected as the most appropriate algorithm. The programming language used to implement this algorithm in code was C# (pronounced C-Sharp). This algorithm described an efficient way of deducing the prime and essential prime implicants. From this algorithm, the implicants could be deduced, the minimum cover could be identified and the step-by-step and final solution could be displayed. Thus it is suitable to meet most of the project's objectives.

The QM method [9], [10] involves two main steps. The first is developing a table to find all the prime implicants via a series of comparisons and second, using the prime implicants found to develop another table to deduce the essential prime implicants. Petrick's method is incorporated at this stage to systematically deduce the essential prime implicants from this table [11].

A separate algorithm was developed to identify the Static 1 and Static 0 hazards [12]. This is discussed in further detail in the section titled "Implementation of the Static Hazards Feature."

This paper discusses an overview of the Karnaugh Mapping tool, followed by a discussion on how the first step of the QM algorithm was implemented to deduce the prime implicants, how the second step of the QM algorithm (Petrick's Method) was implemented to deduce the essential prime implicants, how the Static 1 and Static 0 Hazard features were implemented, then it is discussed how the complete Karnaugh Map Tool was tested by students and their feedback, and lastly, the suggestions for future work to be done.

2. Overview of the Karnaugh Mapping Tool

The Karnaugh Mapping Tool's main purpose is that it can be used in the teaching of the topic of Karnaugh Maps to students. Thus, the tool's graphical user interfaces were designed to be simplistic, allowing for easy navigation. The homepage of the tool is shown in Figure 1 below. From here the student can easily navigate to the 'Start a Karnaugh Map Calculation' page (See Figure 2), the 'About Karnaugh Maps' page or the 'Program Guide' page.The 'About Karnaugh Maps' page details fundamental information on the topic of Karnaugh Mapping and the 'Program Guide' pages has guidelines on how to use the Karnaugh Mapping tool.

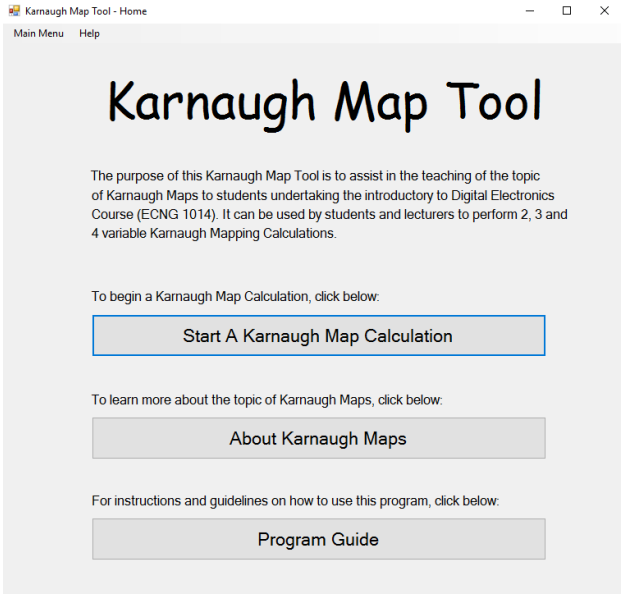

Figure 1: Karnaugh Map Tool Homepage

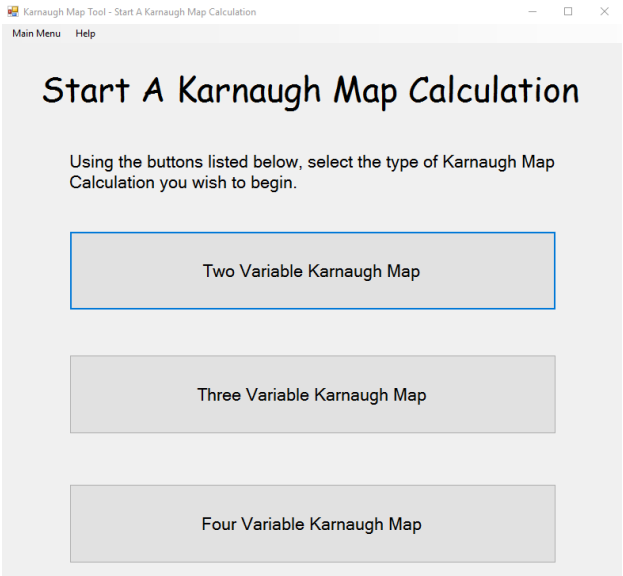

Figure 2: Start a Karnaugh Map Calculation Page

If the user clicks *'Start a Karnaugh Map'* calculation, as seen in Figure 2 above, the user is presented with the option to begin either a 2, 3 or 4 variable Karnaugh Map Calculation. Since each of the pages and the code for the 2, 3 and 4 variable calculations are set up similarly, this paper will only go in-depth in discussing that of the 3 variable Karnaugh Map.

Figure 3 below shows the screen that appears if the user clicks the 'Three Variable Karnaugh Map' button. On this page the user can enter data on the Karnaugh Map in one of two ways. Either by clicking the cells on the Karnaugh Map or the various outputs on the truth table. The values change between "1", "0" and "X" for don't care conditions.

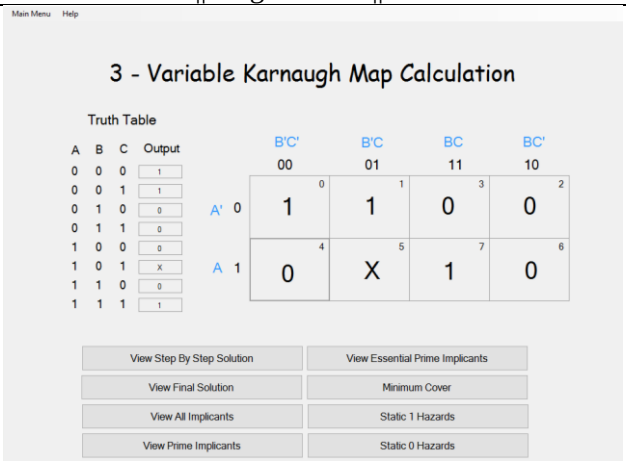

Figure 3: Three Variable Karnaugh Map Calculation Page

When the user has completed entering the values on the Karnaugh Map, they can then select which result they wish to view. For example, if the user wishes to view the prime implicants present, the can click the *'View Prime Implicants'* button beneath the Karnaugh Map and so on. Figure 4 shows the screen the user would see if they were to click *'View Prime Implicants'*.

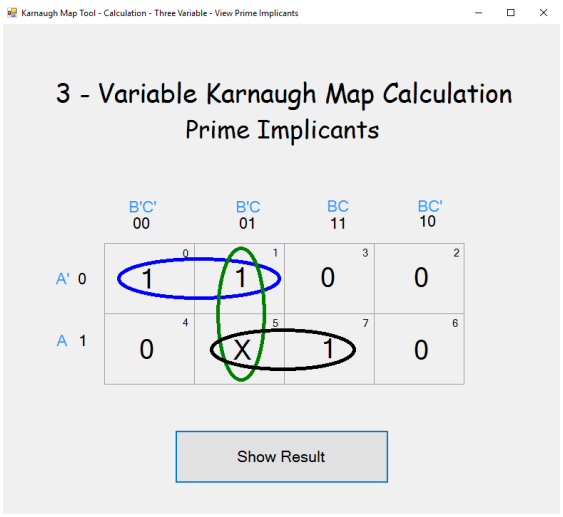

Figure 4:Three Variable Prime Implicants Page

Similar screens are displayed for the other results (Step by Step Solution, Minimum Cover etcetera) the distinction being that the results shown on the Karnaugh Map itself would vary to suit the button clicked by the user.

3. Implementation of First Stage of QM Method (Determining Prime Implicants)

To determine the prime implicants using the QM method, a table is constructed based on the minterms and don't care terms present in the Karnaugh Map (e.g. In Figure 3 the minterms present were 0, 1, 7 and the don't care term present was 5). These terms are converted to their corresponding binary form and sorted in a table based on the number of ones present in each binary term (e.g. the terms in Group 0 have no ones present, terms in Group 1 have a single one present and so on). Table 1 below shows how these terms were sorted.

Table 1: Prime Implicants Table

Group	List 1	List 2	List 3
0	000	00-	
1	001	-01	
2	101	1-1	
3	111		

To implement this in C# code, switch cases were used. The condition of the switch-case being the user input. For example, if the user entered a '1' at position zero in the Karnaugh Map, 000 would be stored in group 0. Similarly, if the user entered a 'X' at position zero in the Karnaugh Map, 000 would be stored in group 0. However if the user entered '0' at position zero in the Karnaugh Map, nothing would be stored in any of the groups as '0' indicates that neither a don't care (X) nor minterm is present. This was repeated in code for all the different positions in the Karnaugh Map, the result being that all the minterms and don't cares entered by the user were correctly sorted.

After sorting, the next step is to compare all consecutive groups. That is, all Group 0 terms are compared with all Group 1 terms, all Group 1 terms are compared with all Group 2 terms and so on. Using Table 1 as reference, under List 2, we see the term 00-. This indicates that terms 000 and 001 were successfully compared. The rule being that, if two terms differ by only one value they can be combined and the differing value replaced with a dash. In terms of the Karnaugh Map, this is saying that a prime implicant can be formed by circling the terms at the zero and one positions. From figure 4, the ones circled in blue represent this. This step is repeated for all the terms present in the table until no further comparison is possible. All list 2 terms also have to be compared with each other, however in this example, no further comparisons are possible hence List 3 remains empty.

In code this was implemented by checking which miterms and don't care terms were present, in which groups they were stored to deduce whether or not they could be successfully combined to form a prime implicant. Figure 5 below shows a code snippet example.

```
//1. Comparision between minterms 0 and 1.
if (Group0List1Unique.Count == 1 & Group1List1AUnique.Count == 1)
{
    //Sucessfully combine Minterms 0 and 1 and store in List 2.
    //Concerning the third variable in the array,Position A=4, Position B=2 and Postion C=1.
    ZeroOneComp[0] = "0";
    ZeroOneComp[1] = "1";
    ZeroOneComp[2] = "1";
    List2.Add(ZeroOneComp);

    //Put a tick mark "t" by 0 and store in List 1 Extension.
    ZeroList1Ext[0] = "0";
    ZeroList1Ext[1] = "t";

    //Put a tick mark "t" by 1 and store in List 1 Extension.
    OneList1Ext[0] = "1";
    OneList1Ext[1] = "t";
}
```

Figure 5: Determining the Prime Implicants Code

It is essentially saying that if '1's are present at position zero (variable named *Group0List1Unique* in the code)and position one (variable named *Group1List1AUnique*) on the Karnaugh Map then the terms can successfully be combined to form a prime implicant.

Similar code was repeated for all other possible comparisons that could be present in the table based on the user input. This ensures that any combination of inputs entered by the user could be compared in the code, thus successfully determining the Prime Implicants present in the Karnaugh Map. Both the 'View Prime Implicants' 'View All Implicants' pages uses the data obtained at this stage to display the respective form of the result to the user.

4. Implementation of Second Stage of QM Method (Petrick's Method – Determining Essential Prime Implicants)

Determining the prime implicants alone however, is not sufficient to deduce the final sum of products (SOP) expression. For this to be calculated, the program then has to determine which are the essential prime implicants, that is, the prime implicants that cover minterms that are not covered by any other prime implicant. In the example given in Figure 4 the blue and black circled prime implicants are essential prime implicants as they cover positions zero and seven respectively which are not covered by any other prime implicant. Whereas the green is not an essential prime implicants, as it covers positions one and five which are both already covered by other prime implicants.

The QM method states that to determine the essential prime implicants, a table has to be created using the prime implicants. This table then has to be analyzed to determine which of the prime implicants present are essential. The essential prime implicant table corresponding to the example given in Figure 4 is shown in Table 2 below. The prime implicants are listed as columns and the minterms as the rows. It should be noted that the don't cares are omitted from this table.

Table 2: Essential Prime Implicants Table

	A'B'	**B'C**	**AC**
0	X		
1	X	X	
7			X

To determine which the essential prime implicants are, we look for the minterm rows with only one 'X' present. In this case, rows 1 and 3. To do this in code, we first have to deduce which prime implicants are contained in the prime implicants list – a list created in the code to store the prime implicants found in the previous step of the QM method described in Section III. When the prime implicants present are deduced, the binary form of each needs to be stored in a new list called *PrimeImplicantsBrinary*. For the example discussed in this paper, the binary form of the prime implicant A'B' would be stored as 00- in the *PrimeImplicantsBinary* list. Following this step, the dash present needs replaced with both a 0 and a 1 giving 001 and 000 now stored in new independent variables.Each of these new variables would then be compared to the binary form of all the minterms present. Whenever there is a match with a minterm when compared an 'X' would be stored in that corresponding minterms list. These steps are imitating the function of the table. When this is completed for all the prime implicants, depending on which miterms list only has one 'X' present, we can determine which the essential prime implicants are.

After the essential prime implicants are deduced, it needs to be determine whether or not the function is covered by the essential prime implicants alone or if other prime implicants need to be included to cover the function. In the case of the example given in Figure 4, the essential prime implicants alone cover the function. As we can see, if only the blue and black circles are present on the map all the minterms i.e. '1's will be covered. This may not be the case in some instances however and a further step needs to be taken to determine which additional prime implicants need to be included to cover the function. This is where Petrick's method is applied.

Petrick's method involves further analyzing of the table to determine the minimum combination of the remaining prime implicants are necessary to cover the function. In Petrick's method, a Boolean expression P is formed which describes all possible solutions of the table. The prime implicants are numbered, P_1 = A'B', P_2 = B'C and P_3 = AC for example. Using these P_i variables, a larger Boolean expression P can be formed, which captures the precise conditions for every row in the table to be covered. For example for row two of the essential prime implicant table (see Table 2) to be covered the expression $P_1 + P_2$ would be written. This would be completed for all the rows of the table forming the larger Boolean expression. For the example given in Figure 4, as we already established, additional prime implicants are not needed to cover this function, thus further explanation using the current example is not valid. However, the Petrick's method would continue to be explained.

After the final Boolean expression is written it may look something like the following example:

$$P = (P_1 + P_2)(P_1 + P_3)(P_2 + P_4)(P_3 + P_5)(P_4 + P_6)(P_5 + P_6) = 1 \quad (1)$$

Using the Boolean algebra rule (X + Y)(X + Z) = X + YZ, as well as the distributive law, the function becomes:

$$P = (P_1 + P_2P_3)(P_4 + P_2P_6)(P_5 + P_3P_6) \quad (2)$$

$$P = (P_1P_4 + P_1P_2P_6 + P_2P_3P_4 + P_2P_3P_6)(P_5 + P_3P_6) \quad (3)$$

$$P = P_1P_4P_5 + P_1P_2P_5P_6 + P_2P_3P_4P_5 + P_2P_3P_5P_6 + P_1P_3P_4P_6 + P_1P_2P_3P_6 + P_2P_3P_4P_6 + P_2P_3P_6 \quad (4)$$

The rule X + XY = X is then applied to remove redundant terms from the logic function producing:

$$P = P_1P_4P_5 + P_1P_2P_5P_6 + P_2P_3P_4P_5 + P_1P_3P_4P_6 + P_2P_3P_6 \quad (5)$$

From this we extract the solution for which the minimum number of prime implicants are used. In the above expression we can either select P_1, P_4 and P_5 or P_2, P_3 and P_6. These in conjunction with the essential prime implicants found will comprise the final sum of product result.

To implement this in code, first, the essential prime implicants were removed from the table and any other minterms they covered. This reduces the table and leaves just the remaining prime implicants that can possibly form the minimum SOP expression. After the table is reduced, the remaining prime implicants in the table needed to be numbered P_1, P_2, P_3 etcetera. For each row, the corresponding expression had to be stored and then these expressions were all combined to form the larger Boolean expression. The program then needed to apply the Boolean algebra rule and the distributive law followed by the X + XY = X rule to calculate the final expanded expression. The program then extracted the solution from the final expanded expression with the least amount of prime implicants present and added these prime implicants to the final SOP. If there are more than one solution with the same number of prime implicants the program will provide all the possible results to the user.

After the additional prime implicants needed to cover the function were determined(if any that is), the final SOP expression was written. The final SOP for the example we have been following thus far, is:

F = A'B' + AC (6)

The final SOP is displayed in a text box at the bottom of the 'View Essential Prime Implicants' page. The 'Final Solution', 'Minimum Cover' and Step by Step Solution' pages also use the data obtained at the end of this stage to present the respective form of the result to the user.

5. Implementation of the First Stage of Static Hazard Feature

A static hazard occurs when a single input variable change should cause no change in the output of a combinational logic circuit, but a short glitch of the incorrect logic level occurs. The problem occurs because real physical implementations of logic functions have finite propagation times which are variable, and if two inputs to a gate should theoretically change simultaneously, one will actually change before the other. There are two types of static hazards:

- Static-1 Hazard: the output is currently 1 and after the inputs change, the output momentarily changes to 0 before settling on 1
- Static-0 Hazard: the output is currently 0 and after the inputs change, the output momentarily changes to 1 before settling on 0

In properly formed two-level AND-OR logic, a Sum Of Products expression will have no static-0 hazards. Conversely, there will be no static-1 hazards in an OR-AND implementation of a Product Of Sums expression.

A static 1 hazard may occur in a two level sum of products (SOP) implementation.A static one hazard can be detected by observing if any two logically adjacent cells with a '1' output are not covered by a common product or implicant, a static hazard can occur when a single input change moves from one cell to the other.

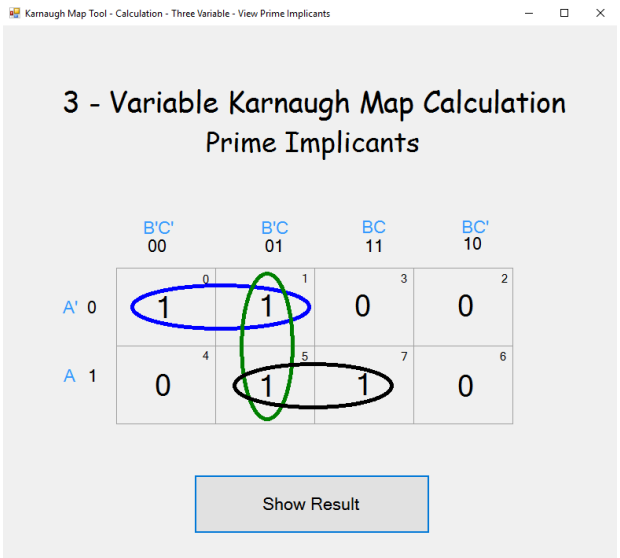

Figure 6: Static 1 Hazard Example

In Figure 6, if the green circled prime implicants is omitted, a Static 1 Hazard can occur. A static 1 hazard can be prevented by adding a product terms so that all pairs of logically adjacent cells with a '1' output have at least one common product covering them. This can be accomplished by using all prime implicants in the SOP form rather than using a minimized SOP form, i.e. leaving the green circled prime implicant expression in the final SOP. This was relatively simple to implement in code. All the possible instances where a Static 1 Hazard could be present in the Karnaugh Map were coded and if a Static 1 Hazzard was indeed present, the group of ones causing the Hazard would be circled in red and their expression would be included in the final SOP to eliminate the hazard.

A static 0 hazard may occur in a two level product of sums (POS) implementation. A static zero hazard can be detected by observing if any two logically adjacent cells with a '0' output are not covered by a common sum. A static hazard can occur when a single input change moves from one cell to the other.

A static 0 hazard can be prevented by adding sum terms so that all pairs of logically adjacent cells with a '0' output have at least one common sum covering them.

A static 0 hazard follows the same basic concept as the static 1 hazard and thus the code implementation is similar.

6. Testing of the Karnaugh Mapping Tool

Upon completion of the implementation of the tool, a group of thirty-four (34) students currently pursuing the degree of Electrical and Computer engineering were given the program to test. A questionnaire was be distributed to gain their feedback on the tool. The most prominent outcome of this survey was that when students were asked whether or not the found the Karnaugh Map Tool to be helpful 100% of the responses said yes. In an open ended follow up question students stated that they were able to verify their answers on Karnaugh Map problems they practiced themselves.

7. Conclusions

The purpose of this Karnaugh Map tool is to improve student performance. From the result of the survey carried out, it was found the 100% of the students that tested the Karnaugh Map Tool found it helpful when studying the topic of Karnaugh Mapping.

Based on the survey carried out and comparison made with existing Karnaugh Map Tools, the following are a list of the suggestions for future work that can be done to improve the Karnaugh Map Tool created:-

- Develop the tool to be capable of solving for more variable Karnaugh Map calculations (at least up to 8),
- Develop the tool to provide the Product of Sums (POS) expression,
- Develop the tool to generate random examples,
- Develop the tool to display a drawing of the final digital circuit to the user, and
- Develop the tool to include video tutorials and/or access to lecture notes in the “About Karnaugh Map” feature.

References

[1] Yizhu Zhao and Yunfeng He. 2012. Some Key Issues of Teaching Reform about Digital Logic. 2012 IEEE Asia-Pacific Services Computing Conference (APSCC), pp. 406 – 409.

[2] Wang Yuan and Liang Zhi-yong. 2012. The application of LabVIEW in the digital logic experiment. 2012 IEEE Symposium on Robotics and Applications (ISRA), pp. 125 - 128, DOI: 10.1109/ISRA.2012.6219137

[3] Arjuna Madanayake, Chamith Wijenayake, Rimesh M. Joshi, Jim Grover, Joan Carletta, Jay Adams;, Tom Hartley and Tokunbo Ogunfunmi. 2012. Teaching freshmen VHDL-based digital design. 2012 IEEE International Symposium on Circuits and Systems, pp. 2701 – 2704.

[4] Pornpimon Chayratsami. 2013. Supplementary laboratory in digital circuit and logic design course for pre-service vocational teacher in Thailand. 2013 IEEE Global Engineering Education Conference (EDUCON), pp. 612 – 617.

[5] A. Alasdoon, P. Prasad, A. Beg and A. Chan, "A Recent Survey of Circuit Design Tools for Teaching", *Proceedings of the World Congress on Engineering and Computer Science*, vol., 2013 [Online]. Available: http://www.iaeng.org/publication/WCECS2013/WCECS2013_pp182-186.pdf. [Accessed: 01- Apr- 2017]

[6] P. W. C. Prasad, Abeer Alsadoon, Azam Beg and Anthony Chan. 2014. Incorporating simulation tools in the teaching of digital logic design. 2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE), pp. 18 – 22.

[7] Zhong Bocheng;, Zizhuo Yang, Yihan Wang and Runcai Huang. 2014. Exploration and practice in teaching of digital logic course. 2014 9th International Conference on Computer Science & Education (ICCSE), pp. 799 – 802.

[8] George, Marcusand Geetam Singh Tomar. 2015. Hardware Design Procedure: Principles and Practices. 2015 Fifth International Conference on Communication Systems and Network Technologies. pp. 834 – 838

[9] M. Osama, "Karnaugh Map Minimizer (Three Variables) - CodeProject", *Codeproject.com*, 2017. [Online]. Available: http://www.codeproject.com/Articles/37031/Karnaugh-Map-Minimizer-Three-Variables. [Accessed: 01- Apr- 2017]

[10] P. Nowick, "The Quine-McCluskey Method", pp. 1-15, 2016 [Online]. Available: http://www.cs.columbia.edu/~cs6861/handouts/quine-mccluskey-handout.pdf. [Accessed: 01- Apr- 2017]

[11] T. ARTICLES, I. ARTICLES, G. ELECTRONICS, C. PROJECTS, E. MICRO, M. SCIENCE, V. Lectures, I. Webinars, C. Library, C. Us and D. Krambeck, "Prime Implicant Simplification Using Petrick's Method", *Allaboutcircuits.com*, 2017. [Online]. Available: http://www.allaboutcircuits.com/technical-articles/prime-implicant-simplification-using-petricks-method/ . [Accessed: 01- Apr- 2017]

[12] "Static Hazards", *Ee.scu.edu*, 2017. [Online]. Available: http://www.ee.scu.edu/classes/2000fall/elen021/supp/stathaz.html. [Accessed: 01- Apr- 2017]

[13] "Digital Logic - Karnaugh Maps", *Facstaff.bucknell.edu*, 2017. [Online]. Available: https://www.facstaff.bucknell.edu/mastascu/eLessonsHTML/Logic/Logic3.html. [Accessed: 01- Apr- 2017]

Author Profile

Marcus Lloyde George received the Bsc degree in Electrical and Computer Engineering from the University of the West Indies, St. Augustine in 2007 and his MPhil degree in Electrical and Computer Engineering from the University of the West Indies, St. Augustine in 2011. Marcus is currently in the examination stage of his PhD degree in Electrical and Computer Engineering from the University of the West Indies, St. Augustine.

Marcus is Chairman, Chief Executive Officer and Founder of the Ultimate Virtual Market Limited Group of Companies which include a range of online-based service-providing companies in the areas of Agriculture, Automotive and Education. He is also the author of several books in the area of Life Foundations and is the author of upcoming books "Expert Mathematics: Strategies and Solutions" and "Digital Electronic Systems – Principles and Practices".

His research engineering interest include the business administration, strategic planning and management, engineering education, formal specification, modelling and verification, field programmable architectures, embedded systems design, intelligent electronic instrumentation, CADs for field programmable architectures, biomedical engineering, network on chip architectures, reconfigurable computing, and information and communication technology (ICT).

Monique Sampson received the Bsc degree in Electrical and Computer Engineering from the University of the West Indies, St. Augustine in 2017 and was a former research student of Marcus Lloyde George.