UNIVERSITY OF NAIROBI

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

BOOLEAN FUNCTION MINIMIZER

PROJECT INDEX: PROJ 022

BY

OWUOR ZEDEKIAH ODOYO

F17/1440/2011

SUPERVISOR: PROF. H. A. OUMA

EXAMINER: MR. COLLINS OMBURA

Project report submitted in partial fulfillment of the requirement for the award of a degree of Bachelor of Science in ELECTRICAL AND ELECTRONICS ENGINEERING of the University of Nairobi 2016

Submitted on:
DECLARATION OF ORIGINALITY

FACULTY/ SCHOOL/ INSTITUTE: Engineering
DEPARTMENT: Electrical and Information Engineering
COURSE NAME: Bachelor of Science in Electrical & Electronics Engineering

NAME OF STUDENT: OWUOR ZEDEKIAH ODOYO
REGISTRATION NUMBER: F17/1440/2011
COLLEGE: Architecture & Engineering
WORK: Boolean Function Minimizer

1) I understand what plagiarism is and I am aware of the university policy in this regard.

2) I declare that this final year project report is my original work and has not been submitted elsewhere for examination, award of a degree or publication. Where other people’s work or my own work has been used, this has properly been acknowledged and referenced in accordance with the University of Nairobi’s requirements.

3) I have not sought or used the services of any professional agencies to produce this work.

4) I have not allowed, and shall not allow anyone to copy my work with the intention of passing it off as his/her own work.

5) I understand that any false claim in respect of this work shall result in disciplinary action, in accordance with University anti-plagiarism policy.

Signature: ……………………… Date: ………………………

Approved by:
Supervisor: Prof. H. A. OUMA
Signature: ……………………… Date……………………..
DEDICATION

To my family for supporting me through my university education and supporting me through everything.
ACKNOWLEDGMENT

I would like to express my gratitude to Prof. H. A. Ouma, who was my supervisor, for his guidance throughout the project. I must particularly thank him for his commitment and unrelenting effort to see me do all the assignments appertaining to this project and meet my deadlines as outlined in my time budget. He was also always available for consultation and gave particularly useful insights concerning the project. I thank him for his advice and I owe him the success of the project.

I would also like to extend my gratitude towards the examiner, Mr. Ombura for taking his time to go through this project documentation and also handling the presentation of the project.

Finally, I would like to thank my friends, classmates and particularly Mark Wahome for his invaluable help especially in designing the user interface and some aspects of C++ which I was not particularly familiar with. With his help I was able to come up with a working a working project.
Table of Contents

DECLARATION OF ORIGINALITY.................................................................ii
DEDICATION .................................................................................................iii
ACKNOWLEDGMENT ..................................................................................iv
LIST OF TABLES ............................................................................................vii
LIST OF FIGURES ...........................................................................................viii
ABSTRACT .......................................................................................................ix

CHAPTER 1: .....................................................................................................1
  GENERAL BACKGROUND ...............................................................................1
  PROBLEM STATEMENT ...............................................................................1
  PROJECT JUSTIFICATION ..........................................................................1
  OBJECTIVES ..............................................................................................2
  SCOPE OF THE PROJECT ...........................................................................2
  PROJECT REPORT ORGANIZATION ..........................................................2
  TIMELINE ...................................................................................................3

CHAPTER 2: LITERATURE REVIEW ...............................................................4
  BOOLEAN ALGEBRA ....................................................................................5
  K-Maps .......................................................................................................6
  REPRESENTATION OF FUNCTIONS ON K-MAPS .....................................11
  SUM-OF-PRODUCT FORM. (SOP FORM) ..................................................12
  PRODUCT-OF-SUM FORM (POS FORM) ...................................................12
  MINIMIZATION USING K-MAPS .............................................................13
  DON’T CARE CONDITIONS ......................................................................14
  QM PROCEDURE .......................................................................................15
  QM PROCEDURE USING DECIMAL NOTATION .......................................19
  PROGRAMMABLE LOGIC ARRAYS (PLAs) .............................................22

CHAPTER 3: SYSTEM DESIGN .....................................................................22
  INPUT STAGE .............................................................................................23
  GROUPING THE MINTERMS DEPENDING ON NUMBER OF 1S IN THE BINARY FORM OF THE MINTERMS ........................................26
  MATCHING .................................................................................................27
  DETERMINATION OF ESSENTIAL PIs ......................................................30
  DETERMINING THE LITERALS FOR EACH ESSENTIAL PI ..................32
  IMPLEMENTATION OF THE REDUCED FUNCTION ................................35

CHAPTER 4: SYSTEM IMPLEMENTATION ..................................................36
  INPUT STAGE .............................................................................................36
Scenario involving minterms. (no don’t care conditions) ......................................................... 38
Scenario involving minterms and don’t care conditions .............................................................. 39
SORTING STAGE .......................................................................................................................... 40
The following codes were used to implement the sorting function: ............................................ 42
Matching ......................................................................................................................................... 43
DETERMINATION OF ESSENTIAL PRIME IMPLICANTS ....................................................... 44
DISPLAY OF THE FINAL MINIMIZED FUNCTION ................................................................. 48
DEVELOPING THE NETLIST ....................................................................................................... 48
CHAPTER 6: RESULTS AND ANALYSIS ................................................................................... 50
WELCOME SCREEN ....................................................................................................................... 50
FORM A: MINIMISATION FOR CASES WHERE THERE ARE NO DON’T CARES. ................. 51
FORM A: MINIMISATION FOR CASES WHERE THERE ARE ‘DON’T CARE MINTERMS’ ............. 55
SAMPLE RESULTS ....................................................................................................................... 58
CHAPTER 7: SUMMARY, CONCLUSION AND RECOMMENDATIONS ..................................... 59
SUMMARY ..................................................................................................................................... 59
Input Stage ..................................................................................................................................... 59
Minimization Stage ....................................................................................................................... 59
Final Stage: ................................................................................................................................... 59
CONCLUSIONS .............................................................................................................................. 59
RECOMMENDATIONS .................................................................................................................. 59
Bibliography ................................................................................................................................. 60
APPENDIXES ............................................................................................................................... 61
Appendix One: Code used in Welcome screen .............................................................................. 61
Appendix Three: Code used in the form for minimization for cases where there are ‘don’t care minterms’ ................................................................................................................... 63
Appendix Three: Code used in the form for minimization for cases where there are ‘don’t care minterms’ ................................................................................................................... 82
# LIST OF TABLES

Table 1: Timeline ..........................................................................................................................3
Table 2: Tables showing AND, OR and complementary operations .............................................5
Table 3: Minterms arranged according to number of 1s ...............................................................16
Table 4: The resultant table after matching minterms in Table 3 ..................................................16
Table 5: Resultant table after matching contents of Table 4 ..........................................................17
Table 6: Table for determination of the essential PIs ...................................................................17
Table 7: Minterms arranged according to number of 1s ...............................................................18
Table 8: The resultant table after matching minterms in Table 7 ..................................................18
Table 9: The resultant table after matching minterms in Table 8 ..................................................18
Table 10: Table for determination of the essential PIs and PIs ......................................................19
Table 11: The numbers are arranged according to the number of 1s in their binary form then paired .................................................................................................................................20
Table 12: Further pairing ..............................................................................................................21
Table 13: No further matching is possible ......................................................................................21
Table 14: Determination of Essential PIs ......................................................................................21
Table 15: Sample results when the Boolean minimizer is used to reduce selected function .........58
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>An example of a Boolean function implemented using logic gates</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Relationship between truth tables and K-Maps</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Relating Venn diagrams and K-Maps</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Three Variable K-Map</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>4-Variable K-Map</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>4-Variable K-Map with block numbering reversed</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>3-Variable K-Map with numbering reversed</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>6-Variable K-Map</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>Representation of functions of K-Map</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>Representation of logic function in SOP form on a K-Map</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>Representation of a logic function POS on a K-Map</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>Minimization using K-Map</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>Don't care conditions</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>Determination of number of variables</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>Flowchart for determination of essential PIs</td>
<td>31</td>
</tr>
<tr>
<td>16</td>
<td>Flowchart for determination of literals of each PIs</td>
<td>33</td>
</tr>
<tr>
<td>17</td>
<td>Flowchart for the generation of the final reduced function</td>
<td>34</td>
</tr>
<tr>
<td>18</td>
<td>Welcome Screen</td>
<td>50</td>
</tr>
<tr>
<td>19</td>
<td>With Don't care form</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>Without don’t care Form</td>
<td>51</td>
</tr>
<tr>
<td>21</td>
<td>Entering Values in the Without Don't care form</td>
<td>51</td>
</tr>
<tr>
<td>22</td>
<td>Dialog box showing error message when a non-integer character is entered</td>
<td>52</td>
</tr>
<tr>
<td>23</td>
<td>Entering the literal to represent the terms in the reduced function</td>
<td>52</td>
</tr>
<tr>
<td>24</td>
<td>Dialog box showing error message when character is not an alphabet</td>
<td>53</td>
</tr>
<tr>
<td>25</td>
<td>Entering the individual minterms</td>
<td>53</td>
</tr>
<tr>
<td>26</td>
<td>Dialog box showing minterms can only be integer characters</td>
<td>54</td>
</tr>
<tr>
<td>27</td>
<td>The result of the minimization process</td>
<td>54</td>
</tr>
<tr>
<td>28</td>
<td>Saving the netlist generated</td>
<td>55</td>
</tr>
<tr>
<td>29</td>
<td>With don’t care Form</td>
<td>56</td>
</tr>
<tr>
<td>30</td>
<td>Error when number of minterms exceeds the maximum possible for the number of variables declared</td>
<td>56</td>
</tr>
<tr>
<td>31</td>
<td>Entering literals</td>
<td>57</td>
</tr>
<tr>
<td>32</td>
<td>Dialog box showing that the minterm entered is outside the range of the possible number of minterms</td>
<td>57</td>
</tr>
<tr>
<td>33</td>
<td>Dialog box showing error when one tries to enter a floating point as a minterm</td>
<td>58</td>
</tr>
</tbody>
</table>
ABSTRACT

Logic circuits are widely used in the world today in industries and manufacture of electronics such as computers. The functionality of logic circuits can be represented using truth tables. Truth tables can be regarded as a pictorial representation of the Boolean function. The number of terms and literals in a Boolean function determine the complexity of a circuit hence the cost. Therefore, it is desirable to come up with the simplest possible logic function to reduce costs and complexity. There are three common ways namely; Boolean algebra, K-Maps and QM method.

Boolean algebra involve using a number of laws and rules to evaluate logic functions. These include the associative law, distributive law, De Morgan’s theorem and convolution theorem. The success of this method depends upon intuition and thus it is prone to errors. It also becomes tedious when you are dealing with logical functions with so many terms.

K-Maps are a graphical representation of the truth tables. K-Maps are suitable for simplification of logical functions of up to 4 variables. They may also be used for 5 or 6 variables. The K-Maps can also be viewed as a modified form of Venn diagrams. They also rely on intuition to reduce logic functions. They are fairly accurate. However, it becomes cumbersome for beyond 4 variables.

Both K-Maps and Boolean algebra have one inherent problem; they rely on intuition. It is therefore very difficult to use to develop a software. There is therefore a need for a standard method of reducing logic functions which does not rely on intuition. The answer was the QM procedure. This procedure involves grouping minterms with the same number of 1s and then comparing each minterm in a group with minterms in the next group. If the difference is a power of 2 the minterms/elements are paired. The matching continues until no more matching is possible. The essential PIs are then determined. The essentials PIs then form the final reduced function. The QM procedure can be used to solve functions of any number of variables which is advantageous.

Hardware implementation of the reduced function may include RAM, ROM, PLA and PAL.
CHAPTER 1: GENERAL BACKGROUND

Most of the modern day, most of the digital devices have integrated circuits which have logic circuits which operate on binary logic (1s and 0s). The logic circuits operation are often represented using Boolean functions and the operation of the circuit represented using truth tables.

It is often desired to find simpler equivalents of the logic circuit. This has the following advantages:

1. A simpler realization of the logic circuits means less components are used and hence cost of designing the circuits is reduced.
2. Since less components are used, there is a reduction in heat dissipation in the circuit which often necessitates the design a cooling system.
3. Increased speed and response time as there is less propagation delay in the logic circuits.

We can therefore see the importance of minimization of the Boolean functions. The current trend in Integrated circuit technology is marked by the desire to fit more components in a smaller space hence the LSIs, VLSIs et cetera. There is therefore need to develop algorithms which are not only used to optimize simple circuits but complex logic circuits which comprises of millions of components.

Boolean function minimization can be performed using several methods such as Boolean algebra, Venn diagrams, Karnaugh maps and the Quine McCluskey or Tabular method.

Boolean algebra is an algebraic system that was develop by George Boole in 1854. It can be defined as algebraic structure defined by a set of elements together with two binary operations. A two valued Boolean algebra was introduced by C. Shannon in 1938. This could be used to represent the switching properties of an electrical circuit. Boolean algebra deals with binary variables together with the logic operations. A function described by an algebraic expression composed of binary variable and logic operations is known as a Boolean function. Boolean algebra is only suitable for smaller simpler circuits with few variables (components) and gets tedious with increased complexity of the Boolean function. It also relies on intuition which means the Boolean function might not always be in its simplest form.

K- MAP is a pictorial form of representation of the truth table of a logic circuit. It is used to do away with the complexity that arises in Boolean algebra due to increased variables. K-MAPS can be implemented for two, three, four variables etc. However, beyond 4 variables the process starts becoming cumbersome. The minimized Boolean function can be expressed in sum-of-products form or products-of-sum form.

PROBLEM STATEMENT:

A Boolean function minimizer is used in optimizing of logic circuits. The problem posed by this project is the development of a software that would enable the reduction of a Boolean function of up to 8 variable functions. The result of this reduction, that is, the minimized function is to be displayed together with its implementation using logic gates.

PROJECT JUSTIFICATION

The project was undertaken to give develop a software that would reduce a given Boolean function to its simplest hence give the smallest logic circuit possible. This is particularly important in the manufacture of integrated circuits where it is desired to put as many logic circuits in the smallest space possible. Each of the elements in an integrated circuits such as logic circuits occupy space and costs money. Thus by minimizing the Boolean function representing the operation of the circuit there is a reduction in the hardware required such as number of logic gates and consequently the costs. There will also be a drop in power dissipation and the speed of response will improve as less
circuit elements means a reduction in propagation delay. Therefore, Boolean function minimizer helps us develop optimized logic circuits which are more efficient.

The Boolean function minimizer is just not meant to display the minimized Boolean function but also to show the implementation of the minimized Boolean function using logic gates. This enhances user friendliness especially for those users who have limited or no background on digital electronics.

The program is to be used for functions of up to 8 variables which are tedious to solve using the common methods such as Venn diagrams, k-maps and Boolean algebra. This will be implemented using Tabular method or even K-maps.

OBJECTIVES
The main objective this objective is to develop a software that performs minimization for a Boolean function consisting of 8 variables and implements the reduced function using logic gates.

The project was carried out with the following specific objectives:

- To develop a software that accepts a Boolean function of up to 8 variables and performs minimization of the Boolean function.
- To develop a user interface that display the minimized Boolean function (output) and its implementation using logic gates.
- To study the various methods that can be used for minimization of logic functions (literature review).
- To find an appropriate method for implementation of the reduced logic functions.
- To design and construct a system that performs Boolean function minimization and implementation.
- To test the system using an appropriate set of data to ensure the objective of the project is achieved.

SCOPE OF THE PROJECT
The project requires the development of a software that performs Boolean function minimization for an up to 8 variables Boolean function. The use of Boolean algebra or Venn diagram is tedious in this case. The two viable options include use of Karnaugh maps and The Quine McCluskey algorithm. The input interface will be designed such that the user can enter the Boolean function to be reduced and choose the form of the minimized equation to be either in sum-of-product form or product-of-sum form. The output interface is meant to show the minimized function and its implementation in form of a logic circuit. There are two options in generating the logic circuit of the minimized function: the logic circuit can be generated by the Boolean function minimizer software which would imply that we create a library consisting of the logic gates and devices. The second option is to generate set of codes which if keyed in to another program generates the logic circuit.

PROJECT REPORT ORGANIZATION
The project will be organized into 5 chapters as follows:

Chapter 1:
This covers the background (area of project), objectives of the project and its scope. The methodology is also described.

Chapter 2:
This chapter covers the literature review which is based on the background information of the problem statement. The chapter includes all material studied in relation to the project. The various methods of Boolean function minimization and their merits and demerits are studied here.
Chapter 3:  
This is where the design process of the Boolean function is described showing the various modules and their implementation.

Chapter 4:  
The chapter involves the actual construction and testing of the system

Chapter 5:  
This chapter covers the discussions, conclusions and recommendations for further development.

**TIMELINE**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Topic</th>
<th>Subtopic</th>
<th>Duration</th>
<th>Week</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 1: Timeline*
CHAPTER 2: LITERATURE REVIEW

A Boolean data type is a data type that accepts only two possible outcomes (binary). These could be true or false, on or off, 1 or 0 and many others. The algebra that deals with binary variables and logic operations is referred to as Boolean algebra. A Boolean function can be described as a logic expression consisting of binary variables (0, 1) and logic operation symbols such as ‘AND’ and ‘OR’. A Boolean function can be represented using a truth table. The number of rows in the truth table will be equivalent to 2^n where n is the number of variables in the Boolean function.

The Boolean function can be implemented into a logic circuit using logic gates which are connected depending on the logic function itself. Each product will require a gate for implementation and each variable (literal) acts as an input to a gate. A literal can be defined as a single variable in a product term. An example of a Boolean function implemented using logic gates is shown below. 

F=ABC + ABC’ + A’BC’


We can therefore see that by manipulation of the Boolean function using suitable method such as Boolean algebra we are able to obtain a simpler Boolean function which implies a reduction in the number of gates and inputs. This is called Boolean function minimization. The reduction in complexity and number of gates implies a reduction of producing the logic circuit.

Why the need for Boolean function minimization?

How complex a logic circuit is depends on the number of logic gates. Therefore, the complexity of a logic gate is usually dependent on the number of inputs. Since a logic circuit is an implementation of a Boolean function, the number of inputs to each gate can be reduced by reducing the number of terms (literals) in the function hence reducing the complexity of the circuit. (circuit minimization)

It is important to understand that merely reducing the number of terms in a Boolean function is not always the only criteria in circuit optimization. There are other considerations such as:

- Regularity of structure. This is from the fabrication point of view.
- Minimization of the area covered by the logic circuit on the silicon wafer.
Thus merely minimizing the Boolean function is not a guarantee of a lesser complex circuit during implementation of the logic circuit. The logic programmable circuits are implemented using the programmable logic arrays. (PLAs)
However, circuit minimization by reducing the number of literals by reducing the number of literal in the SOP form of the function has inherent advantages which include:
- Reduction in costs of hardware
- Increased speed and reduced time as less components imply less propagation delay.
- Less heat dissipation
There are several method of circuit minimization which include:
1. Boolean algebra
2. Karnaugh maps (K-Maps)
3. Tabular method also referred to as the Quine-McCluskey procedure (QM procedure)

**BOOLEAN ALGEBRA**

Boolean algebra was developed in the year 1854 by George Boole. Later in 1938, a two valued Boolean algebra called the switched algebra was developed by C.E. Shannon. These enabled the representation of bistable electrical circuits (logic circuit).
The switched algebra can be described as a set of two elements \( B = \{0, 1\} \) which obey the following laws (Huntington postulates):

a) The structure is always closed which implies that the results will always be a 0 or 1.
b) For the ‘AND’, ‘OR’ and complementary operations:
\[
\begin{align*}
1 + 0 &= 0 + 1 = 1 \\
1 + 1 &= 1 \\
1 \cdot 1 &= 1 \\
1 \cdot 0 &= 0 \cdot 1 = 0 \\
1’ &= 0 \quad \text{and} \quad 0’ = 1
\end{align*}
\]
This can be shown by the truth tables below:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( x + y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( x \cdot y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x )</th>
<th>( x’ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
c) Commutative laws
   \[ X + Y = Y + X \]
   \[ XY = YX \]

d) Associative laws
   \[ X + (Y + Z) = (X + Y) + Z \]
   \[ X \cdot (YZ) = (XY) \cdot Z \]
e) Distributive law
   \[ X (Y + Z) = XY + XZ \]

f) De Morgan’s Theorem
   \[ (X + Y + Z)' = X'Y'Z' \]
\[ (XYZ)' = X' + Y' + Z' \]

g) Absorption Theorem
   \[ X + XY' = X \text{ and } (X + Y') = X \]
h) Involution Theorem
   \[ (X')' = X \]

An example of minimization carried out using Boolean algebra is as follows:
\[ F = X'Y'Z + X'YZ + XY' \]
\[ = X'Z (Y' + Y) + XY' \]
\[ = X'Z + XY' \]

Minimization of Boolean functions using the algebraic method has several shortcomings:
   i. It is tedious and prone to errors
   ii. The success of the method depends on intuition. It depends on one’s ability to recognize the
       appropriate rules, theorems or postulate which is not always obvious.
   iii. There is lack of a standard set of rules to guide one during the minimization. Therefore,
       approach may differ.

To overcome the above problems we use the graphical method of Boolean function methods namely
K-Maps and QM procedure. Both methods involve tables hence are mechanical. K-Maps are
suitable for minimization of functions with up to 5-6 variables. QM procedure on the other hand can
be used for minimization of functions of any variable and is easily implemented with a suitable
programming language.

It is important to note that that using both methods the final reduced function may differ depending
on the choices made during the minimization process. However, all minimized functions having the
same number of literals (terms) are regarded to have the same complexity hence any of them can be
used during implementation.

**K-Maps**

This method was developed by Maurice Karnaugh in 1953. It is most suited for simplification of
logic functions with up to 4 variables. It can be used for functions having 5 or 6 variables even
though this is rather tedious.

The K-Map can be regarded as a graphical representation of a truth table. It is made up of squares
each representing a minterm of the logic function to be minimized. The Boolean function is
recognized by the squares representing minterms which have been included in the Boolean function.
The K-Maps can also be regarded as modified Venn diagrams. Consider the Venn diagram for variables A and B. (Two variables)

**Venn Diagrams**

- Each set of minterms represents a Boolean function.
- Examples:

  \[
  \begin{align*}
  \{a, b, a', b'\} & \rightarrow a.b + a.b' = a.(b+b') = a \\
  \{a', b, a, b\} & \rightarrow a'.b + a.b = (a'+a).b = b \\
  \{a, b\} & \rightarrow a.b \\
  \{a, b, a', b', a', b\} & \rightarrow a.b + a.b' + a'.b = a + b \\
  {} & \rightarrow 0 \\
  \{a'.b', a.b, a.b', a'.b\} & \rightarrow 1
  \end{align*}
  \]

Recall
The four areas in the Venn diagram correspond to the squares in the K-Map. We can show the relationship between K-Maps and the Venn diagram as follows:

Look at the second column in the 2 variable K-Map

A = AB’ + AB
This can be seen in the Venn diagram. If we shade the areas represented by AB’ and AB the resultant shaded area is equivalent to A.
This can be verified using Boolean algebra as follows:
AB’ + AB

=A (B’ + B) Recall B’ + B = 1
Hence,
=A (1)
=A

Now if we consider first column in the 2 variable K-Map

A’ = A'B' + A'B
This can be seen in the Venn diagram. If we shade the areas represented by A’B’ and A’B the resultant shaded area is equivalent to A’.
This can be verified using Boolean algebra as follows:
A’B’ + A'B

=A’ (B’ + B) Recall B’ + B = 1
Hence,
=A’ (1)
=A’

Now if we consider the first row in the 2 variable K-Map

B’ = A’B’ + AB’
This can be seen in the Venn diagram. If we shade the areas represented by A’B’ and AB’ the resultant shaded area is equivalent to B’.
This can be verified using Boolean algebra as follows:
A’B’ + AB’

=B’ (A’ + A) Recall A’ + A = 1
Hence,
=B’ (1)
=B’

Lastly, if we look at the second row in the 2 variable K-Map

B= A'B + AB
This can be seen in the Venn diagram. If we shade the areas represented by AB’ and AB the resultant shaded area is equivalent to A.
This can be verified using Boolean algebra as follows:
A’B’ + AB

=B (A’ + A) Recall A’ + A = 1
Hence,
=B (1)
=B

The result is shown in the K-Map in diagram which represents the usual form.
Three Variable K-Map
The three variable K-Map is shown below

![Three Variable K-Map](http://www.experts mind.com/CMSImages/323_3%20variable%20K%20maps.png)

It consists of 8 blocks ($2^3$) which corresponds to 8 minterms.
The three variable K-Map is designed such that A corresponds to two most right hand columns, B corresponds to the two middle columns and C corresponds to the second row. This is shown in figure (a).
Therefore each variable corresponds to four blocks in the K-Map. Figure (b) shows the block numbers and allocation of variables.

Four Variable K-Map.
Note that an n-variable K-Map contains $2^n$ blocks. Each of these blocks correspond to a minterm of the n-variable function. Also, there will be exactly $2n$ blocks in the K-Map corresponding to each variable in the logic function.
The K-Map has a unique property that only one variable changes as one moves from one block to the adjacent block on the K-Map. This is due to the gray code representation of the K-Map.
Two minterms are said to be logically adjacent if they differ in only one variable. This means that the differing variable occurs in true form in one variable and complemented form in the other variables while the other variables appear in the same form in both minterms.
The K-Maps therefore transform the logical adjacency into physical adjacency of the minterms. This implies that minterms represented physically adjacent to each other on a K-Map are also logically adjacent to each other.
The blocks on the last row of a 4-variable K-Map are adjacent to the corresponding blocks on the last row. The same applies to the first and last columns. The 4-variable K-Map can thus be viewed as a toroid. The top and bottom row are brought together forming a cylinder whose ends are also brought together hence forming the toroid. This property is important during the minimization process.
The diagram below shows the 4-variable K-Map

![4-Variable K-Map](http://www.experts mind.com/CMSImages/323_3%20variable%20K%20maps.png)
The K-Maps below show another block numbering. These forms are valid as physical adjacency and logical adjacency are maintained.

Figure 6: 4-Variable K-Map with block numbering reversed [5]

Figure 7: 3-Variable K-Map with numbering reversed [6]

Five-Variable K-Map
This is shown in the figure 3.4 below

http://verticalhorizons.in/wp-content/uploads/2012/12/5_variable_k_map.bmp

Notice that on the left-hand map A = 0 and on the right-hand map A = 1. Due to this the planes are held one over the other.
The two blocks occupy the same position on the plane and are adjacent to each other. The two maps are treated as two planes. The two plane are superimposed on the other. Blocks occupying the same position on each map are said to be logically adjacent.

Six-Variable K-Map
It consists of 4 planes. This is shown in the figure below.
Each plane is adjacent to the plane to its left (or right) and the plane below or above it. The adjacent planes should be held over each other to bring the logically adjacent blocks into physical adjacency.

By comparing the various K-Maps we can deduce the following:
- The degree of adjacency is directly proportional to the number of variables. This means that each block in an n-variable K-Map has n blocks adjacent to it.
- It is difficult to represent an adjacency degree of more than 6 on a K-Map. This can be attributed to the fact that a cube has only six faces or sides.

Beyond 6 variables the K-Maps cease to be useful hence tabular methods become more useful at this point.

**REPRESENTATION OF FUNCTIONS ON K-MAPS.**

On a Venn diagram the areas representing the logic functions are shaded. However, in a K-Map, a value 0 or 1 is inserted on each block of a K-Map depending on the value of the function. The blocks whose number correspond to a minterm in the logic expression have a value of 1 whereas the other blocks will have a value of 0. This is illustrated by the following example:

The three variable K-Map below is used to represent the function

\[ F = \overline{A'B'C'} + AB'C' + ABC + AB'C \]
**SUM-OF-PRODUCT FORM (SOP FORM)**

A logic expression is in the sum-of-product form if it is given as an ‘OR’ operation of logic function that are in ‘AND’ form. Consider the expression below for a 3-variable K-Map

\[ Y = A'B'C' + A'B'C + A'B'C + AB'C + ABC + ABC + ABC + ABC \]

Each term in the logic expression is referred to as a minterm. The minterms are defined as follows:

- \( m_0 = A'B'C' \)
- \( m_1 = A'B'C \)
- \( m_2 = A'B'C \)
- \( m_3 = A'B'C \)
- \( m_4 = AB'C' \)
- \( m_5 = AB'C \)
- \( m_6 = ABC' \)
- \( m_7 = ABC \)

\[ Y = m_0 + m_1 + m_2 + m_3 + m_4 + m_5 + m_6 + m_7 \]

The above relation is expressed as follows

\[ Y = \sum m (0, 1, 2, 3, 4, 5, 6, 7) \]

To plot a function given in SOP form, derive the list of minterms and then plot the minterms on a K-Map.

Alternatively one can intersect the areas of the K-Map corresponding to each product term. This is illustrated by the following example:

\[ Y = \sum m (0, 2, 3, 4, 8, 9, 10, 11, 13, 14) \]

![K-Map](image)

*Figure 11: Representation of logic function in SOP form on a K-Map*

**PRODUCT-OF-SUM FORM (POS FORM)**

A logic expression is in the sum-of-product form if it is given as an ‘AND’ operation of logic functions that are in ‘AND’ form. Consider the expression below for a 3-variable K-Map

\[ Y = (A + B + C) (A + B' + C) (A' + B + C) (A' + B' + C) (A' + B' + C) \]

Each term in the logic expression is referred to as a maxterm. The maxterms are defined as follows:

- \( M_0 = A + B + C \)
- \( M_1 = A + B + C' \)
- \( M_2 = A + B' + C \)
- \( M_3 = A + B' + C' \)
- \( M_4 = A' + B + C \)
- \( M_5 = A' + B' + C \)
- \( M_6 = A' + B' + C' \)
- \( M_7 = A' + B' + C' \)

\[ Y = M_0 M_1 M_2 M_3 M_4 M_5 M_6 M_7 \]

The above relation is expressed as follows

\[ Y = \prod M (0, 1, 2, 3, 4, 5, 6, 7) \]

To plot a function given in POS form, derive the list of maxterms and then plot the maxterms on a K-Map. This is similar to plotting SOP form except 0s are used instead of 1s.

*This is illustrated by the following example:*
MINIMIZATION USING K-Maps

Recall that two functions on a K-Map that are physically adjacent are also logically adjacent. Two adjacent 1s in a K-Map can be combined.

Steps

i. Plot the logic expression on a K-Map. Form groups of adjacent pairs of 1s. If a 1 is not adjacent to any other 1 in the K-Map it forms a group by itself.

ii. Try to combine the groups formed in the first step to the largest group possible. It is important to note that all groups must have a number of 1s equivalent to a power of 2.

iii. When grouping ensure that all 1s are covered at least once. The same 1 can be part of several books.

iv. Choose the smallest number of groups that can cover all the 1s in the K-Map.

v. Find the product term of each of these groups (AND).

vi. Add (OR) the product terms to get the minimized Boolean function.

The following terms are important during minimization:

**Implicant**: This is a group of 1, 2, 4, 8… \(2^n\) 1s.

**Prime Implicant**: A product term that is not covered by any other term of the Boolean function. It is likely to be a term in the final minimized function. Each of the product terms obtained by grouping 1s in a K-Map is a prime Implicant.

**Essential Prime Implicant**: This is a PI that covers a minterm that is not covered by any other PI and thus must form part of the final minimized function.

Once essential PIs have been selected, one or more nonessential PIs are selected to cover the remaining minterms. During this selection, care is taken to ensure that PIs corresponding to larger groups are selected hence a much simpler function.

**Illustration of Minimization**

\[ f(A, B, C, D) = \Pi M(2, 6, 8, 9, 10, 11, 14) \]
\[ f(wxyz) = \Sigma m(0, 1, 5, 7, 8, 10, 14, 15) \]

\[
\begin{array}{c|cccc}
  \text{wx} & 00 & 01 & 11 & 10 \\
  \hline
  \text{yz}\backslash \text{wx} & 00 & 1 & 0 & 0 \\
 01 & 1 & 1 & 0 & 0 \\
11 & 0 & 1 & 1 & 0 \\
10 & 0 & 0 & 1 & 1 \\
\end{array}
\]

**Solution 1**

\[
\begin{array}{c|cccc}
  \text{wx} & 00 & 01 & 11 & 10 \\
  \hline
  \text{yz}\backslash \text{wx} & 00 & 1 & 0 & 0 \\
01 & 1 & 1 & 0 & 0 \\
11 & 0 & 1 & 1 & 0 \\
10 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[ f = \overline{wz} \overline{y} + \overline{wz} \overline{y} + \overline{wz} \overline{y} \]

**Solution 2**

\[
\begin{array}{c|cccc}
  \text{wx} & 00 & 01 & 11 & 10 \\
  \hline
  \text{yz}\backslash \text{wx} & 00 & 1 & 0 & 0 \\
01 & 1 & 1 & 0 & 0 \\
11 & 0 & 1 & 1 & 0 \\
10 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[ f = \overline{w} \overline{x} \overline{z} + \overline{w} \overline{x} \overline{z} + \overline{w} \overline{x} \overline{z} + \overline{w} \overline{x} \overline{z} \]

*Figure 13: Minimization using K-Map [9]*

[https://www.cs.tau.ac.il/~nin/Courses/mivne98/design/Image54.gif](https://www.cs.tau.ac.il/~nin/Courses/mivne98/design/Image54.gif)

**DON'T CARE CONDITIONS**

*Figure 14: Don't care conditions [10]*

QM PROCEDURE
The QM procedure involves the use of the Quine–McCluskey algorithm for Boolean function minimization. It is often referred to as tabular method or sometimes method of prime implicants. It was developed by W.V. Quine and Edward J. McCluskey in 1956. It is functionally similar to Karnaugh mapping. The tabular form used in this procedure makes it easy to implement using a suitable programming language. The QM procedure also gives a deterministic way to check that the minimal form of a Boolean function has been reached.

The QM procedure involves two steps:
1. Finding all prime implicants of the function.
2. Use those prime implicants in a prime implicant chart to find the essential prime implicants of the function, as well as other prime implicants that are necessary to cover the function.

Steps
Step 1: Express each individual minterm (and don’t cares if included in the function to be minimized) of the given logic expression to its binary equivalent. Group the minterms (and don’t cares if included in the function to be minimized) with equivalent number of 1s on the same group to form a table. Start with minterms with zero 1s, then one 1s, then two 1s and so on.

Step 2: Compare each minterm in the n group with each minterm in the n+1 group. If the two minterms are the same in every position but one, place a tick to the right of both minterms to show that they have been paired and covered. Enter the newly formed number in the next column of a new table. The new number has the similar literals plus an ‘X’ in the position of the differing literals.

Step 3: Repeat the process in step 2 above and form a second table. Repeating the process until no more matching or pairing can be done.

Step 4: The terms in the final table plus any unchecked term (without tick) in any of the tables are the prime implicants.

Step 5: We obtain the essential prime implicants. Form a table of the prime implicants and the minterms (columns). Put a tick along each row for each minterm covered by a prime implicant. If a minterm is covered by only one prime implicant then it is an essential prime implicant. Check if the obtained essential prime implicants cover all the other minterms. If so then the final minimized function is the sum of these essential prime implicants. If not so then select suitable prime implicants to cover the uncovered minterms. The sum of these prime implicants and essential prime implicants gives the final minimized in each case.

Note: If the function to be minimized contained don’t care values these are omitted in step 5 where essential prime implicants and other prime implicants are to be obtained.

The following two example illustrate the use of QM procedure:

Example 1
Y (A, B, C, D) = ∑ (0, 1, 3, 7, 8, 9, 11, 15)

Step 1
First we express all the minterms to their binary equivalent
0 = 0000
1 = 0001
3 = 0011
7 = 0111
8 = 1000
9 = 1001
11 = 1011
15 = 1111

Minterms with the same number of 1s are then grouped together

<table>
<thead>
<tr>
<th>Groups</th>
<th>Minterms</th>
<th>A B C D</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1 1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 2
We then compare the minterms in the n group and n+1 group in the table obtained in step 1. If the minterms vary by only one they are considered matched and an X is placed in place of the varying variable. We place a check mark beside every matched minterm.

The result is the new table shown below:

<table>
<thead>
<tr>
<th>Group</th>
<th>Matched pairs</th>
<th>Binary representation</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>m₀ - m₁</td>
<td>000X</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>m₀ - m₈</td>
<td>X000</td>
<td>✓</td>
</tr>
<tr>
<td>1</td>
<td>m₁ - m₃</td>
<td>00X1</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>m₁ - m₉</td>
<td>X001</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>m₈ - m₉</td>
<td>100X</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>m₃ - m₇</td>
<td>0X11</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>m₃ - m₁₁</td>
<td>X011</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>m₉ - m₁₁</td>
<td>10X1</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>m₇ - m₁₅</td>
<td>X111</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>m₁₁ - m₁₅</td>
<td>1X11</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4: The resultant table after matching minterms in Table 3

Step 3
We now compare the minterms in the n group and n+1 group in the table obtained in step 2. If the minterms vary by only one they are considered matched and an X is placed in place of the varying variable. We place a check mark beside every matched minterm.

The result is the new table shown below:

<table>
<thead>
<tr>
<th>Group</th>
<th>Matched pairs</th>
<th>Binary representation</th>
<th>PRIME IMPLICANTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>m₀ - m₁ - m₈ - m₉</td>
<td>X00X</td>
<td>B’C’</td>
</tr>
<tr>
<td></td>
<td>m₀ - m₈ - m₁ . m₀</td>
<td>X00X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>m₁ - m₃ - m₉ - m₁₁</td>
<td>X0X1</td>
<td>B’D</td>
</tr>
</tbody>
</table>
Notice that further matching is not possible hence this table gives us the prime implicants.  

**Step 4**

To determine the essential prime implicants we construct the table below. Place an x for every minterm covered by a prime implicant. If a minterm column has a single ‘x’ (marked red) then the associated prime implicant is considered essential. Notice that that the two essential prime implicants cover all the minterms. The two thus form the final minimized function.

<table>
<thead>
<tr>
<th>Pls</th>
<th>Minterms covered</th>
<th>0</th>
<th>1</th>
<th>3</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>B’C’</td>
<td>0, 1, 8, 9</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B’D</td>
<td>1, 3, 9, 11</td>
<td>X</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>3, 7, 11, 15</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The final minimized function is $Y = B’C’ + CD$

**Example 2**

**Step 1**

$Y (A, B, C, D) = \sum m(4, 6, 9, 10, 11, 13) + \sum d(2, 12, 15)$

<table>
<thead>
<tr>
<th>Minterms with the same number of 1s are then grouped together</th>
<th>A B C D</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2 m4</td>
<td>0010</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>0100</td>
<td>✓</td>
</tr>
<tr>
<td>Groups 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M6 M9 M10 m12</td>
<td>0110</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>1010</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>1001</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>1100</td>
<td>✓</td>
</tr>
</tbody>
</table>
Step 2

We then compare the minterms in the n group and n+1 group in the table obtained in step 1. If the minterms vary by only one they are considered matched and an X is placed in place of the varying variable. We place a check mark beside every matched minterm. The result is the new table shown below:

<table>
<thead>
<tr>
<th>Groups</th>
<th>Minterms</th>
<th>A B C D</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$m_2 - m_6$</td>
<td>$m_2 - m_{10}$</td>
<td>$m_2 - m_6$</td>
</tr>
<tr>
<td>1</td>
<td>$m_9 - m_{11}$</td>
<td>$m_9 - m_{13}$</td>
<td>$m_{10} - m_{11}$</td>
</tr>
<tr>
<td>2</td>
<td>$m_{11} - m_{15}$</td>
<td>$m_{13} - m_{15}$</td>
<td>$11X1$</td>
</tr>
</tbody>
</table>

Table 7: Minterms arranged according to number of 1s

Step 3

We now compare the minterms in the n group and n+1 group in the table obtained in step 2. If the minterms vary by only one they are considered matched and an X is placed in place of the varying variable. We place a check mark beside every matched minterm. The result is the new table shown below:

<table>
<thead>
<tr>
<th>Groups</th>
<th>Minterms</th>
<th>A B C D</th>
<th>Prime Implicant</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$m_9 - m_{11} - m_{13} - m_{15}$</td>
<td>$m_9 - m_{13} - m_{11} - m_{15}$</td>
<td>$1XX1$</td>
</tr>
</tbody>
</table>

Table 8: The resultant table after matching minterms in Table 7

Step 4

Notice that matching is only possible for the checked terms giving us a prime implicant of AD. The unchecked terms in table in step 2 also are also prime implicants and are candidates for the final minimized function.

The resultant prime implicants thus are as follows:

$A’CD’$, $B’CD’$, $A’BD’$, $BC’D’$, $AB’C$, $ABC’$, $AD$

Step 4

To find the essential Prime Implicants:

<table>
<thead>
<tr>
<th>Pls</th>
<th>Minterms</th>
<th>4</th>
<th>6</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To determine the essential prime implicants we construct the table below. Place an x for every minterm covered by a prime implicant. If a minterm column has a single ‘x’ (marked red) then the associated prime implicant is considered essential. Notice that the essential PI AD covers the minterms 9, 11 and 13. We then select two other prime implicants to cover the remaining minterms. In our case AB’C to cover 10 and A’BD’. The final minimized function is therefore

\[ \text{Y} = \text{AD} + \text{AB’C} + \text{A’BD’} \]

**QM PROCEDURE USING DECIMAL NOTATION**

**Step 1:**
The minterms are grouped according to the number of 1s in their binary representation in the decimal format.

**Step 2:**
Compare each minterm in the nth group with larger minterms in the n+1 group down. If they differ by a power of 2 then they pair-off. If the minterms pair check both of them and form a second table consisting of the minterms paired and the decimal difference of the corresponding minterms in the bracket, i.e. \( m_x, m_y (y-x) \).

**Step 3:**
Compare each element in the nth group in the new table with elements of the n+1 group and select numbers that have the same numbers in parenthesis. If the lowest minterm number in the nth group of the table differs by a power of 2 with the corresponding number then they combine.

**Step 4:**
Form a second table by all four minterms in the combined pair followed the differences in the parentheses i.e. the previous value (the difference) and the new difference.

**Step 5:**
Repeat step 4 until no more matching is possible.

**Step 6:**
All the unchecked elements form the prime implicants. Select the common literals from each prime implicant by comparison.

**Step 7:**
We then develop a cover chart to determine the essential prime implicants. Test if the prime implicants cover all the minterms. If so the minimized function is the sum of the essential prime implicants. If not so find the prime implicant which covers the rest of the minterms. The sum of this prime implicants and the essential prime implicants gives the reduced function.

In cases where there are no prime implicants, find the least number of minterms that can cover all the minterms. The sum of these prime implicants gives the reduced function. The following is an example to illustrate this method.

**Example 3**
\[ \text{Y} (A, B, C, D) = \sum (0, 5, 6, 7, 9, 10, 13, 14, 15) \]

**Step 1**
\[ Y(A, B, C, D) = \sum m(0, 5, 6, 7, 9, 10, 13, 14, 15) \]

<table>
<thead>
<tr>
<th>Groups</th>
<th>Minterms</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5, 6, 9, 10</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>2</td>
<td>7, 13, 14</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Table 11: The numbers are arranged according to the number of 1s in their binary form then paired*

**Step 2**

<table>
<thead>
<tr>
<th>Minterm pairs</th>
<th>Differences</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>5, 7</td>
<td>2</td>
<td>✓</td>
</tr>
<tr>
<td>5, 13</td>
<td>8</td>
<td>✓</td>
</tr>
<tr>
<td>6, 7</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>6, 14</td>
<td>8</td>
<td>✓</td>
</tr>
<tr>
<td>9, 13</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10, 14</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7, 15</td>
<td>8</td>
<td>✓</td>
</tr>
<tr>
<td>13, 15</td>
<td>2</td>
<td>✓</td>
</tr>
<tr>
<td>14, 15</td>
<td>1</td>
<td>✓</td>
</tr>
</tbody>
</table>
### Table 12: Further pairing

#### Step 3:

<table>
<thead>
<tr>
<th>Minterm pairs</th>
<th>Differences</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,7-13,15,15</td>
<td>2,8</td>
<td></td>
</tr>
<tr>
<td>5, 13-7,15</td>
<td>2,8</td>
<td></td>
</tr>
<tr>
<td>6, 7 - 14,15</td>
<td>1,8</td>
<td></td>
</tr>
<tr>
<td>6, 14 – 7,15</td>
<td>1,8</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 13: No further matching is possible

#### Step 6
To determine the essential prime implicants

<table>
<thead>
<tr>
<th>Elements/ 0 5 6 7 9 10 13 14 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>9,13</td>
</tr>
<tr>
<td>10,14</td>
</tr>
<tr>
<td>5,7-13,15</td>
</tr>
<tr>
<td>6,14-7,15</td>
</tr>
</tbody>
</table>

Table 14: Determination of Essential Pls

All the prime implicants represented by the elements are essential as indicated by the red cells. The elements cover the minterms hence all form part of the reduced function.

#### Step 7
0 = 0000 = A'B'C'D'
9, 13 (4) = 9, 13 (2^2)
= 1 0 0 1
+ X
1 X 0 1
= A'CD
10, 14 (4) = 10, 14 (2^2)
= 1010
+ X
1 X 10
= ACD'
5, 7-13, 15 (2, 8) = 5, 7-13, 15 (2^1,2^3)
= 0 1 0 1
+ X X
X 1 X 1
= BD
The final minimized function is thus:

\[ Y = A'B'C'D' + AC'D + ACD' + BD + BC \]

**PROGRAMMABLE LOGIC ARRAYS (PLAs)**

This is a programmable logic device with a programmable AND gates and programmable OR gates. It is used in the implementation of a reduced Boolean function. The array of AND gates can be programmed to produce any product term of the input variable. The array of OR gates is used to provide the SOP form of the minimized Boolean function to be implemented. To implement a Boolean function using PLAs we first reduce the Boolean function using an appropriate method. In our case we will use the QM procedure. A fuse table or a PLA programmable table is then generated. The fuse table can then be used to program the PLA.

PLAs can be classified as:

- **Mask programmable**
  This is programmed by the manufacturer and cannot be altered by the user.

- **Field programmable**
  This can be programmed by the user by means of a commercial programmable hardware kit.

**CHAPTER 3: SYSTEM DESIGN**

The operation of the system can be summarized by the following block diagram:
The Boolean function minimizer is meant to perform function for logic function of up to 8 inputs.
variables. This means that the number of variables is important in determining the maximum number of minterms and hence the range of valid minterms. For our case the variables should not exceed 8.

The maximum number of terms in this case is given by $2^n$. For example if number of variables $n$ is equal to 8 the maximum number of terms possible will be $256$ ($2^8$). The individual minterms will vary from 0-$2^n-1$. In the case of 8 variables, it will be from 0-255.

The following program flowchart will demonstrate the working of the input stage.

![Program Flowchart](image)

**Figure 15**: Determination of number of variables

We know that the maximum possible number of variables is 8. If a user enter a figure greater than 8 the program prints “Enter a number from 1-8” on the screen then asks for a new number of variables. A similar process is repeated when the number of minterms is entered. If the number of minterms exceeds $2^n$ then the program prints “Enter a number from 1 to $2^n$” and asks for the
number of minterms to be reentered.

Once the number of minterms in the logical function to be minimized is known the system proceeds to ask for the individual minterms. If the minterm is greater than $2^n - 1$ the system returns a message “Enter minterm between 0-2^n-1” and asks for a minterm. The minterm entered is stored in its allocated memory location. The minterm entered is then converted to its binary equivalent which is also stored in an allocated memory location. Once this is done the system requests for another minterm. The system loops until the number of the minterms is equal to the initially defined number, that is, for four minterms the system loops four times.
GROUPING THE MINTERMS DEPENDING ON NUMBER OF 1S IN THE BINARY FORM OF THE MINTERMS.

Start

Consider the binary equivalent of the first minterm

Find sum of the digits
Sum = sum of the digits

Store corresponding minterm in the Sum 1s table (BinSum)

Consider the binary equivalent of the next minterm

Have all the minterms been considered?

Stop
The next stage is the grouping of minterms according to the number of 1s in their binary equivalent. This can be done using by simply adding the digits in the binary equivalent of minterm. If the sum of the digits is 0 then the corresponding minterm is stored in the table for 0 1s. If the sum is 1 then the corresponding minterm is stored in its corresponding. If sum is 2 it can be stored in the table for 2 1s and so on. It is important to note that the number of variables has a bearing on the maximum possible number of minterms in each group. This is in accordance with permutation principle. An example is the case of a 4 variables where the total number of minterms possible is 16. We note that:

The maximum number of minterms with 0 1s is 1.
The maximum number of minterms with 1 1s is 4.
The maximum number of minterms with 2 1s is 6.
The maximum number of minterms with 3 1s is 4.
The maximum number of minterms with 4 1s is 1.

The software is to be designed for up to 8 variables. For an 8 variables consideration it follows that:

The maximum number of minterms with 0 1s is 1.
The maximum number of minterms with 1 1s is 8.
The maximum number of minterms with 2 1s is 28.
The maximum number of minterms with 3 1s is 56.
The maximum number of minterms with 4 1s is 70.
The maximum number of minterms with 5 1s is 56.
The maximum number of minterms with 6 1s is 28.
The maximum number of minterms with 7 1s is 8.
The maximum number of minterms with 8 1s is 1.

The process is repeated for all minterms. To increase the efficiency of the program we introduce variables to help us know the number of minterms in each group. This is particularly important during matching process.

**MATCHING**

In this stage we compare each element in a group with all the elements in the next group. Pairing off is only done if the element in next group is larger than elements in the current group and the difference is a power of 2. The elements which satisfy the above two conditions are paired off and checked to indicate so. The difference is stored as it will be important in determining the prime implicants later. The process is repeated until each of the elements in current group has been compared with all elements in the next group. The process is then repeated for the next set of groups until all the groups have been compared.

Once all the groups have been compared we look for unchecked elements. These are stored in the prime implicants. An integer variable is used to keep track of the number of elements in the prime implicant table.

Matching is repeated for the next set of groups created after the initial matching process. Matching continues until no further matching is possible.
Start

Define a variable Count and set the Count to zero
Read minterms in n group and n+1 groups

Read the first minterm in the n group and first minterm in the n+1 group

Is minterm in n+1 > nth group minterm?

Yes

Find difference

Is difference a power of 2?

Yes

Store pair and difference in a new group
Check the minterms paired
Count=Count +1

No

Read the next minterm in the n+1 group
Retain the current minterm in the nth group

Have all the minterms in the n+1 group been considered?

Yes

Stop

No

Have all minterms in the nth group been considered?

No

Read the next minterm in the n+1 group
Retain the current minterm in the nth group

Yes

Read the next minterm in the n+1 group
Retain the current minterm in the nth group

Store count
Reset count
Count=0
Start

Define Pi_count
Set Pi_count = 0

Read all groups and their respective minterms

Consider first group

Consider first minterm

Read next minterm/pair

Is it checked?

Have all minterms/pairs been considered?

Store in the PI table
Pi_count= Pi_count + 1

Have all minterms/pairs been considered?

No

Consider next group

Read next minterm/pair

Yes

No

Stop
DETERMINATION OF ESSENTIAL PIs

Start

 Declare the cover table. The columns should be equal to the number of prime implicants and the rows should be equal to the number of implicants
 Define epi_num

Read the first prime implicant
 Set epi_num = 0

Consider the next prime implicant

Check the column corresponding to the minterms contained in the prime implicant

Have all the prime implicants been considered?

Yes

Consider the first column
 Add all the cells in the column

No

Consider the next column
 Add all the cells in the column

Is the sum 1?

Yes

Look up the cell with 1 in the column (checked)

Store the associated Prime implicant in the essential prime implicant table
 epi_num = epi_num + 1

No

Have all the columns been considered?

Yes

R
In the determining of essential prime implicants we develop a cover chart. The columns of the cover chart is equal to the number of minterms. The rows are equal to the number of prime implicants. For each of the prime implicants (rows) check the cells of its associated with it. This can be done by essentially placing a 1 each cell in the row with an associated minterm. The process is repeated for all the prime implicants. To determine the essential prime implicants we add all the cells in each column. If the sum is one check the cell containing 1 in the column then record the associated prime implicant in the essential prime implicant table. Repeat the process for all the columns.

Once the essential components have been determined we check if they cover all the minterms. There are three possible scenarios:

- The essential prime implicants cover all the minterms. In these case the essential prime implicants form the final reduced function.
- The essential prime implicants don’t cover all the minterms. There is need to find the smallest number of prime implicants that can cover the remaining minterms. These can be stored in the essential prime implicant table for ease in developing the final reduced function.
- The third case is when there is no essential prime implicant. Here we need to find the smallest set of prime implicants that can cover the remaining minterms. These are stored in the prime implicant table.
DETERMINING THE LITERALS FOR EACH ESSENTIAL PI

Express all differences associated with each essential PI in power 2 form. The positions correspond to don’t care digits in the binary form. Express the lowest term in each pair in binary form (8 variables). The binary form for each minterm had already been determined when grouping the minterms according to number of 1s. Thus we only need to look up the values from where they were stored. Then replace all digits of the binary equivalent of the minterm represented by the power two form of differences with ‘X’ (don’t care conditions). In the final representation we ignore the terms with ‘X’. The other terms in the binary form are written as they are or complemented depending on whether it’s represented by a 0 or 1 respectively. In case an essential prime implicant is a minterm. We just convert to its binary equivalents. The literals are then chosen with their corresponding form which may or may not be complemented depending on whether it’s represented by a 0 or 1 respectively. Once all the essential PIs have been expressed in their binary form, their sum is printed to give the final reduced function. The following program flowchart can be used to express this process.
Figure 17: Flowchart for determination of literals of each PI

- **Is \( d = 1 \)?**
  - Place a don’t care condition X on the LSB (first bit) of the binary form of the first minterm in the pair
- **Is \( d = 2 \)?**
  - Place a don’t care condition X on the second bit of the binary form of the first minterm in the pair
- **Is \( d = 4 \)?**
  - Place a don’t care condition X on the third bit of the binary form of the first minterm in the pair
- **Is \( d = 8 \)?**
  - Place a don’t care condition X on the fourth bit of the binary form of the first minterm in the pair
- **Is \( d = 16 \)?**
  - Place a don’t care condition X on the fifth bit of the binary form of the first minterm in the pair
- **Is \( d = 32 \)?**
  - Place a don’t care condition X on the sixth bit of the binary form of the first minterm in the pair
- **Is \( d = 64 \)?**
  - Place a don’t care condition X on the seventh bit of the binary form of the first minterm in the pair
- **Is \( d = 128 \)?**
  - Place a don’t care condition X on the MSB (eighth bit) of the binary form of the first minterm in the pair

Have we considered all the differences?

V (Yes) -> X (No)

Y
To get the final reduced we first create a reference table as follows:

<table>
<thead>
<tr>
<th></th>
<th>MSB</th>
<th>7th Bit</th>
<th>6th Bit</th>
<th>5th Bit</th>
<th>4th Bit</th>
<th>3rd Bit</th>
<th>2nd Bit</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A’</td>
<td>B’</td>
<td>C’</td>
<td>D’</td>
<td>E’</td>
<td>F’</td>
<td>G’</td>
<td>H’</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
</tbody>
</table>

We can then look up the literals of each essential prime implicant in the table. If a literal is represented by a zero look up the corresponding literal in the second row. If a literal is represented by a 1 look up its corresponding alphabetical literal in the third row. If the literal is ‘X’ (don’t care condition) then ignore. This gives the term representing an essential prime implicant. The same process is repeated for the other essential prime implicants. The sum of the terms obtained gives the final reduced function.

![Flowchart for the generation of the final reduced function](image)
IMPLEMENTATION OF THE REDUCED FUNCTION
CHAPTER 4: SYSTEM IMPLEMENTATION

INPUT STAGE

Start

Declare variables int n_var, n_min, n, i, total_min, smin, smin_limit;

Enter number of variables

Enter n_var

total_min = 2^n_var

Enter number of minterms

Enter n_min

Declare MIN[n_min][2]

Is n_min < 0 or > total_min n?

Initialize n = 0

Enter the literals starting from MSB

Enter lit[n]
n = n + 1

Is n < n_var

The number of minterms should be between 0 and...
Set i=0

Enter minterm

Enter MIN[i][0]

Is MIN[i][0]>0 and <total_min?

i=i+1

Is i<n_min?

Do we have don’t care conditions?

Declare int n_minOr, dc_num, dc_trans, bol_dc;

How many don’t care minterms are there?

Enter dc_num

Declare DC[dc_num] Set i=0

Enter don’t care minterms

Enter DC[i]
There are two possible scenarios for Boolean function minimization which include:

- Minimization of functions with minterms only.
- Minimization of functions composed of minterms and don’t care.

For both scenarios the program first requests for the number of variables. This is used to determine the maximum number of minterms and the range of possible individual minterms. The program then requests for the number of minterms (this excludes don’t care minterms) and asks the user to enter the individual minterms. This is the end of the first stage of input for the second scenario. For the second scenario the program then asks if there are don’t care conditions and if so how many. The individual don’t care minterms are then keyed in. Care is taken to ensure that the sum of the minterms and don’t care conditions do not exceed the maximum number of minterms. The don’t care minterms are then transferred to the table holding the other minterms.

**The above flowchart was implemented by the following codes:**

```c++
Scenario involving minterms. (no don’t care conditions)
#include <iostream>
#include <string>
using namespace std;
int main (){
```

![Flowchart](image-url)
```c++
int n_var, n_min, n, total_min, smin, umin_limit;
cout << "Enter number of variables\n";
cin >> n_var;
total_min = 1;
for (smin = 0; smin < n_var; smin++) {
    total_min = 2 * total_min;
}
char lit[n_var];
re_minnum:
cout << "Enter number of minterms\n";
cin >> n_min;
if (n_min < 0 || n_min > total_min) {
    cout << "The maximum number of minterms is ";
    cout << total_min << endl;
    goto re_minnum;
}
umin_limit = total_min - 1;
for (n = 0; n < n_var; n++) {
    cout << "Enter the letters you want to use to represent the final reduced function starting from MSB\n";
    cin >> lit[n];
}
int MIN[n_min][2];
for (i = 0; i < n_min; i++) {
    re_again:
    cout << "Enter Minterm\n";
    cin >> MIN[i][0];
    if (MIN[i][0] < 0 || MIN[i][0] > umin_limit) {
        cout << "The minterms lie between 0 to ";
        cout << umin_limit << endl;
        goto re_again;
    }
}

Scenario involving minterms and don’t care conditions

#include <iostream>
#include <string>
using namespace std;

int main () {
    int n_var, n_min, n, total_min, smin, umin_limit;
cout << "Enter number of variables\n";
cin >> n_var;
total_min = 1;
for (smin = 0; smin < n_var; smin++) {
    total_min = 2 * total_min;
}
char lit[n_var];
re_minnum:
cout << "Enter number of minterms excluding don’t care minterms\n";
cin >> n_min;
if (n_min < 0 || n_min > total_min) {
    cout << "The maximum number of minterms is ";
    cout << total_min << endl;
    goto re_minnum;
}
umin_limit = total_min - 1;
for (n = 0; n < n_var; n++) {
    cout << "Enter the letters you want to use to represent the final reduced function starting from MSB\n";
    cin >> lit[n];
}
}
int MIN[n_min][2];
for(i=0;i<n_min;i++){
    re_again:
    cout<<"Enter Minterm\n";
    cin>>MIN[i][0];
    if(MIN[i][0]<0||MIN[i][0]>umin_limit){
        cout<<"The minterms lie between 0 to ";
        cout<<umin_limit<<endl;
        goto re_again;
    }
}
//Dealing with don't cares
int n_minOr,dc_num,dc_trans,bol_dc,max_dc;
n_minOr=n_min;
cout<<"Do we have any don't care minterms(if Yes type 1 and if NO type 2)\n";
cin>>bol_dc;
if(bol_dc==1){
    cout<<"Presence of don't care conditions\n";
    RE_DC:
    cout<<"How many don't care minterms are there\n";
    cin>>dc_num;
    max_dc=n_min + dc_num;
    if(max_dc>total_min){
        goto RE_DC;
    }
    int M_DC[dc_num];
    for(i=0;i<dc_num;i++){
        re_DC:
        cout<<"Enter DON'T CARE Minterm\n";
        cin>>M_DC[i];
        if(M_DC[i]<0||M_DC[i]>umin_limit){
            cout<<"The minterms lie between 0 to ";
            cout<<umin_limit<<endl;
            goto re_DC;
        }
    }
    n_min=n_min+dc_num;
    dc_trans=0;
    for(i=n_minOr;i<n_min;i++){
        MIN[i][0]=M_DC[dc_trans];
        dc_trans=dc_trans+1;
    }
}

SORTING STAGE
This involves converting each individual minterm into its binary form and then counting the number of ones the number of ones in each case. The numbers of 1s for binary equivalent of each minterm is then stored in a corresponding cell. After the minterms have had the number of 1s in their binary equivalents determined we proceed to sort the minterms according to the number of 1s starting with zero. The maximum number of 1s is always equal to the number of variables. The number of variables in each groups is noted as frequency. The following

![Diagram](image-url)
Initialize y = n\_var

z = y - 1;  
p1 = tz \mod 2;  
Dec[s][z] = p1;  
tz = tz / 2;

Is y > 0?

Is s < n\_min?

J = 0

Find sum  
MIN[j][1] = sum

Is j < n\_min?

Check how many minterms have 0  
1s and then group them record the  
number of minterms for each group

Stop
The following codes were used to implement the sorting function:

//Conversion to Binary Form
int Dec[n_min][n_var];
int tz,s,q,y,p1;
for(s=0;s<n_min;s++){
    tz=MIN[s][0];
    for(y=n_var;y>0;y--){
        z=y-1;
        p1=tz%2;
        Dec[s][z]=p1;
        tz=tz/2;
    }
}

//sorting depending on number of 1s
int j,k,l,sum;
for(j=0;j<n_min;j++){
    sum=0;
    for(k=0;k<n_var;k++){
        if(Dec[j][k]==1){
            sum=sum+1;
        }
    }
    MIN[j][1]=sum;
}

int Table1[n_min][2];
int Freq[n_var+1][2];
n_check=0;
t05=Freq[0][0];
recheck:
n_freq=0;
for(l1=0;l1<n_min;l1++){  
    if(MIN[l1][1]==n_check){
        Table1[n_t1][0]=MIN[l1][0];
        n_t1=n_t1+1;
        n_freq=n_freq+1;
    }
}  
Freq[m_freq][0]=n_freq;
++m_freq;
if(n_check<n_var||n_check==n_var){
    n_check=n_check+1;
goto recheck;
}  
Freq[0][1]=Freq[0][0];
t05=Freq[0][0];
if(m_freq-1>1){
    for(t02=1;t02<m_freq-1;t02++){
        t04=t02-1;
        t03=Freq[t04][1]+Freq[t02][0];
        Freq[t02][1]=t03;
    }
}
Matching
Here we apply the concepts of frequency and cumulative frequency. The frequency represents the number of minterms in each group. Knowing the frequencies and cumulative frequencies helps in matching of adjacent groups: The following codes shows implementation of the matching process:

```c
//Matching
int m_pair,count_groups,tst,m1,m2,m3,t,pair_each,x,y1,t2,diff;
t=t789,pzt,N_PI,t10,t11,t12,t13,t14,t15,t16,P_freq;
P_freq=0;
int MID[256][256];
int FINAL[256][256];
int FINAL_FREQ[256][2];
int FINAL_diff[256][256];
int Table_diff[256][256];
int Mid_freq[256][256];
int CHECK[256][2];
int PI[256][256],PI_Ndiff[256],PI_diff[256][256]:int PI_Freq[256][2];
N_PI=0;
m_pair=0;
count_groups=0;
for(t=0;t<4;t++)
{tst=Freq[t][1];
m1=tst-Freq[t][0];
m2=Freq[t+1][1];
m3=m2-Freq[t+1][0];
pair_each=0;
for(x=m1;x<tst;x++)
{t1=Table1[x][0];
 for(y1=m3;y1<m2;y1++)
 {t2=Table1[y1][0];
  diff=t2-t1;
  if(diff==1||diff==2||diff==4||diff==8||diff==16||diff==32||diff==64||diff==128){
   MID[m_pair][0]=t1;
   MID[m_pair][1]=t2;
   Table_diff[m_pair][0]=diff;
   m_pair=m_pair+1;
   pair_each=pair_each+1;
   Table1[x][1]=1;
   Table1[y1][1]=1;
  }
  }
  pair_each=0;
}
N_freq[count_groups][0]=pair_each;
count_groups=count_groups+1;
}
for(pzt=0;pzt<n_min;pzt++)
{if(Table1[pzt][1]!=1)
 {PI[N_PI][0]=pzt;
  PI_Ndiff[N_PI]=0;
  N_PI=N_PI+1;
  P_freq=P_freq+1;
  }
  }
```
DETERMINATION OF ESSENTIAL PRIME IMPLICANTS

We develop a cover chart to determine the essential prime implicants. The rows should be 1 more than the number of prime implicants. The columns represent each minterms and the maximum number of columns is equal to the total minterms. We initialize all the cells of the cover chart to have zeros then assign 1 for every minterm covered by each prime implicants to the correspond. The find the sum for each column and store in the last cell of each column. If the sum in the column is 1 then mark the corresponding prime implicant. These are the first essential prime implicants. We test if the essential PIs determined at this stage covers all minterms then the PIs are used to form the final minimized function.

If the essential prime implicants do not cover all the minterms then we use the columns of the uncovered minterms to form a table. We then find the sum of each row. Take the row and mark it as an essential PI then check the minterms covered by this PI. Check if all the minterms are covered. If true then we have found the terms of the reduced function. If it’s not true then reassign values corresponding to the minterms covered then find the sum of the columns again. Repeat the above procedure.

The code below was used to implement the process above:

```c
//Developing cover chart
int COVER_CHART[N_PI+1][n_min+1];
int COVER_CHECK[n_min],j6,j7,j8,tpz,tpz1,j9,j10;
for(j6=0;j6<N_PI+1;j6++){
    for(j7=0;j7<n_min;j7++){
        COVER_CHART[j6][j7]=0;
    }
}
int NOT[N_PI];
for(j6=0;j6<N_PI;j6++){
    tps=PI_Ndiff[j6];
    tps1=COVER_NTERMS[tps];
    NOT[j6]=tps1;
}
int tps2,tps3,tps4;
for(j6=0;j6<N_PI;j6++){
    tps=NOT[j6];
    for(tps1=0;tps1<tps;tps1++){
        tps2=PI[j6][tps1];
        for(tps3=0;tps3<n_min;tps3++){
            tps4=MIN[tps3][0];
            if(tps2==tps4)
                COVER_CHART[j6][tps3]=1;
        }
    }
}
```
int N_sum;
for(j7=0;j7<n_min;j7++){
    N_sum=0;
    for(j6=0;j6<N_PI;j6++){
        N_sum=N_sum+COVER_CHART[j6][j7];
    }
    COVER_CHART[N_PI][j7]=N_sum;
}
//initialize min check
for(j7=0;j7<n_min;j7++)
    COVER_CHECK[j7]=0;
//check each individual PI to determine essential pi
int EPI_check[N_PI];
for(j7=0;j7<N_PI;j7++)
    EPI_check[j7]=0;
for(tps=0;tps<n_min;tps++)
    tps1=COVER_CHART[N_PI][tps];
    if(tps1==1){
        for (j6=0;j6<N_PI;j6++)
            if(COVER_CHART[j6][tps]==1){
                EPI_check[j6]=1;
            }
    }
//test if all minterms are covered
for(j9=0;j9<N_PI;j9++)
    if(EPI_check[j9]==1){
        j6=NOT[j9];
        for(j7=0;j7<j6;j7++)
            tps=PI[j9][j7];
        for(j8=0;j8<n_min;j8++)
            if(tps==MIN[j8][0]){  
                COVER_CHECK[j8]=1;
            }
    }
int N_sum1;
N_sum1=0;
for(j6=0;j6<n_min;j6++)
    if(COVER_CHECK[j6]==1){
        N_sum1=N_sum1+1;
    }
}
cout<<N_sum1<<endl;
int epi;
epi=0;
int EPI[N_PI][n_var];
if(N_sum1==n_min){
    for(j6=0;j6<N_PI;j6++)
        if(EPI_check[j6]==1){
            tps=PI[j6][0];
            for(j7=n_var;j7>0;j7--){
                tps1=j7-1;
                tps2=tps%2;
            }
        }
}
```c
EPI[epi][tps1]=tps2;
    tps=tps/2;
    epi=epi+1;
}

//INSERTING THE DONT CARE CONDITIONS
int p_dcare,tps5,j80;
j80=0;

if(N_sum1==n_min){
    for(j6=0;j6<N_PI;j6++){
        if(EPI_check[j6]==1){
            tps=PI_Ndiff[j6];
            if(tps>0){
                tps1=2;
                tps2=tps1-1;
                for(j7=0;j7<tps;j7++){
                    p_dcare=PI[j6][tps2]-PI[j6][0];
                    cout<<p_dcare<<endl;
                    tps2=tps2*2;
                    tps3=n_var-1;
                    tps4=1;
                    for(j8=0;j8<n_var;j8++){
                        if(p_dcare==tps4){
                            EPI[j80][tps3]=2;
                        }
                    }
                    tps3=tps3-1;
                    tps4=tps4*2;
                }
            }
        }
        j80=j80+1;
    }
    if(N_sum1==n_min){
        for(j6=0;j6<N_PI;j6++){
            if(EPI_check[j6]==1){
                tps=PI_Ndiff[j6];
                if(tps>0){
                    tps1=2;
                    tps2=tps1-1;
                    for(j7=0;j7<tps;j7++){
                        p_dcare=PI[j6][tps2]-PI[j6][0];
                        cout<<p_dcare<<endl;
                        tps2=tps2*2;
                        tps3=n_var-1;
                        tps4=1;
                        for(j8=0;j8<n_var;j8++){
                            if(p_dcare==tps4){
                                EPI[j80][tps3]=2;
                            }
                        }
                        tps3=tps3-1;
                        tps4=tps4*2;
                    }
                }
        }
        j80=j80+1;
    }

    //CASES WHERE THE ESSENTIAL PI s do not cover all the minterms
    int uncovered,uc,uc_sum;
    uc=0;
    if(N_sum1==n_min){
        uncovered=n_min-N_sum1;
        int UNC[N_PI][uncovered+1];
        for(j6=0;j6<n_min;j6++){
            if(COVER_CHECK[j6]==0){
                for(j7=0;j7<N_PI;j7++){
                    UNC[j7][uc]=COVER_CHART[j7][j6];
                }
                uc=uc+1;
            }
        }
        //Determining other essential prime implicants
        //First develop a table for the unchecked minterms
        int Table[uncovered][3];
        int uc2;
        for(j6=0;j6<n_min;j6++){
            if(COVER_CHECK[j6]==0){
                Table[uc2][0]=MIN[j6][0];
                Table[uc2][1]=j6;
                uc2=uc2+1;
            }
        }
        recheck_pi:
        //find sum of columns
```
for(j6=0;j6<N_PI;j6++){
    uc_sum=0;
    for(j7=0;j7<uc;j7++){
        uc_sum=uc_sum+UNC[j6][j7];
    }
    UNC[j6][uncovered]=uc_sum;
}
//Find largest sum in the UNC LAST COLUMN
int t_largest,tl;
t_largest=UNC[0][uncovered];
 tl=0;
for(j6=1;j6<N_PI;j6++){
    if(t_largest<UNC[j6][uncovered]){
        t_largest=UNC[j6][uncovered];
        tl=j6;
    }
}
EPI_check[tl]=1;
tps=NOT[t_largest];
cout<<tps<<endl;
for(j7=0;j7<tps;j7++){
    tps1=PI[tl][j7];
    for(j8=0;j8<n_min;j8++){
        if(tps1==MIN[j8][0]){
            COVER_CHECK[j8]=1;
        }
    }
}
int N_sum2;
N_sum2=COVER_CHECK[0];
for(j6=1;j6<n_min;j6++){
    N_sum2=N_sum2+COVER_CHECK[j6];
}cout<<N_sum2<<endl;
if(N_sum2==n_min){
    for(j6=0;j6<N_PI;j6++){
        if(EPI_check[j6]==1){
            tps=PI[j6][0];
            for(j7=n_var;j7>0;j7--){
                tps1=j7-1;
                tps2=tps%2;
                EPI[epi][tps1]=tps2;
                tps=tps/2;
            }
            epi=epi+1;
        }
    }
}
for(j6=0;j6<N_PI;j6++){
    if(EPI_check[j6]==1){
        tps=PI_Ndiff[j6];
        if(tps>0){
            tps1=2;
            tps2=tps1-1;
            for(j7=0;j7<tps;j7++){
                pdcare=PI[j6][tps2]-PI[j6][0];
                tps2=tps2*2;
                tps3=n_var-1;
                tps4=1;
                for(j8=0;j8<n_var;j8++){
                    if(pdcare==tps4){
                        EPI[j80][tps3]=2;
                    }
                    tps3=tps3-1;
                    tps4=tps4*2;
                }
            }
        }
    }
}
DISPLAY OF THE FINAL MINIMIZED FUNCTION
Here we consider the essential PI table. Depending on the bit position a literal (1) or a complement of the literal (0) is printed. The don’t care conditions are represented by 2 hence will not be considered.

//DISPLAYING THE MINIMIZED FUNCTION
cout<<"THE FINAL REDUCED FUNCTION IS\n";
for(j6=0;j6<epi;j6++)
for(j7=0;j7<n_var;j7++)
if(EPI[j6][j7]==1) {
    cout<<lit[j7];
    if(EPI[j6][j7]==0) {
        cout<<lit[j7];
        cout<<"'\n";
    }
}
if(j6<epi-1){
    cout<<" + ";
}

DEVELOPING THE NETLIST
cout<<"THE IMPLEMENTATION OF THE REDUCED FUNCTION IS GIVEN BY THE NETLIST BELOW\n";
cout<<"\n";
cout<<" IS SCHEMATIC DESCRIPTION FORMAT 8.0\n";
cout<<"=======================================\n";
cout<<"Design: zade.pdsprj\n";
cout<<"Doc. no.: <NONE>\n";
cout<<"Revision: <NONE>\n";
cout<<"Author: <NONE>\n";
cout<<"Created: ";
char date[9];
_strdate(date);
cout << date << endl;
cout<<"Modified: ";
_strdate(date);
cout << date << endl;
cout<<"\n";
cout<<"*PROPERTIES,0\n";
cout<<"\n";
cout<<"*MODELDEFS,0\n";
cout<<"\n";
cout<<"*PARTLIST,2\n";
cout<<"\n";
cout<<"*NETLIST,7\n";
int IMP_TABLE[epi];
int IMP;
IMP=0;
for(j=0;j<epi;j++){
    for(k=0;k<n_var;k++){
        if(EPI[j][k]==0||EPI[j][k]==1){
            IMP=IMP+1;
        }
    }
    IMP_TABLE[j]=IMP;
    IMP=0;
}
int T_IMP,T_IMP1,T_IMP2,T_IMP3,j_count;
j_count=0;
T_IMP1=0;
T_IMP2=IMP_TABLE[0]+1;
int j1;
for(j=0;j<epi;j++){
    cout<<"#0000";cout<<j1;cout<<",2\n";
    T_IMP=IMP_TABLE[j];
    for(k=0;k<T_IMP;k++){
        T_IMP1=T_IMP1+1;
        cout<<"U1";cout<<" ,IP, ";cout<<T_IMP1<<endl;
    }
    j1=j1+1;
    cout<<"\n";
    cout<<"#0000";cout<<j1;cout<<",2\n";
    cout<<"U1";cout<<" ,OP, ";cout<<T_IMP1<<endl;
    cout<<"U2";cout<<" ,IP, ";cout<<"D";cout<<j_count<<endl;
    j_count= j_count +1;
    if (j==0){
        T_IMP1=T_IMP+epi;
    }
    j1=j1+1;
    T_IMP2=T_IMP2+1;
cout<<"\n";
CHAPTER 6: RESULTS AND ANALYSIS

WELCOME SCREEN
When the user opens the program the welcome screen below is displayed.

Figure 19: Welcome Screen

The form is basically a welcome screen with three pushbuttons, namely Exit, Without Don’t care and With Don’t care. Clicking on the exit causes the program to close.
Clicking the “With Don’t care” button causes Form B to open which allows us to perform minimization for logical functions which contain don’t care minterms.

Figure 20: With Don’t care form

Clicking the “Without Don’t care” button causes Form A to open which allows us to perform minimization for logical functions which do not contain don’t care minterms.
FORM A: MINIMISATION FOR CASES WHERE THERE ARE NO DON’T CARES.

When we press the “Without Don’t care” button the Without Don’t care Form opens.
The form has two text boxes for entering the number of variables and the number of minterms.

Both are declared as integers and hence only integers can be entered. Any attempt to enter any non-integer value is detected as an error and one is requested to enter fresh details.
Once the details have been entered press the next button on the form. The system will request for the user to enter the literals which will be used to represent the final reduced function.

The textbox only allows letters of the alphabet both small case and uppercase. If you enter any other details a message dialog box pops which tells you to enter a letter of the alphabet. The number of letters required will be equal to the number of variables.
As soon as the required number of letters are entered the program now requests for the individual minterms using a dialog box.

The minterms should be integers. The minimum integer value for minterm is 0 and the maximum minterm value is dependent on the number of variables.
The maximum value is always less than the two raised to the power of the number of variables. Entering an integer outside this range causes an error message to be displayed giving the range of value of minterms expected. Also if one enters an abnormal data such as a floating number an error message is displayed and the user is required to reenter the values. Once the values have been entered correctly the final reduced function is displayed in the textbox captioned “Obtain reduced function”.

The user will then have an option of saving the netlist developed.
One then presses the return button to return to the original welcome screen.

**FORM A: MINIMISATION FOR CASES WHERE THERE ARE ‘DON’T CARE MINTERMS’**.

In the home screen we also have the “With Don’t care button”. When we press the “Without Don’t care” button the Form A opens. The form has two text boxes for entering the number of variables and the number of minterms. Both are declared as integers and hence only integers can be entered. Any attempt to enter any non-integer value is detected as an error and one is requested to enter fresh details.

---

*Figure 29: Saving the netlist generated*
The user then clicks the check box to confirm the presence of don’t care minterms. The don’t care minterms are also have to be integers and the range depends on the number of variables. The total number of minterms (don’t care and normal) should not exceed $2^{\text{number of variables}}$ otherwise a dialog box pops ups signaling that there are too many minterms.

Once the details have been entered press the next button on the form. The system will request for the user to enter the literals which will be used to represent the final reduced function. The textbox only allows letters of the alphabet both small case and uppercase. If you enter any other details a message dialog box pops which tells you to enter a letter of the
alphabet. The number of letters required will be equal to the number of variables.

As soon as the required number of letters are entered the program now requests for the individual minterms (both normal and don’t care minterms) using a dialog box. The minterms should be integers. The minimum integer value for minterm is 0 and the maximum minterm value is dependent on the number of variables. The maximum value is always less than the two raised to the power of the number of variables. Entering an integer outside this range causes an error message to be displayed giving the range of value of minterms expected.

Also if one enters an abnormal data such as a floating number an error message is displayed and the
user is required to reenter the values.

Figure 34: Dialog box showing error when one tries to enter a floating point as a minterm

Once the values have been entered correctly the final reduced function is displayed in the textbox captioned “Obtain reduced function”. The user will then have an option of saving the netlist developed. One then presses the return button to return to the original welcome screen.

SAMPLE RESULTS
The following results were obtained for the various functions obtained.

<table>
<thead>
<tr>
<th>Boolean Function to be reduced</th>
<th>Reduced Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $Y (A, B, C, D) = \sum (0, 1, 3, 7, 8, 9, 11, 15)$</td>
<td>$B'C' + CD$</td>
</tr>
<tr>
<td>2 $Y (A,B,C,D) = \sum m(4,6,9,10,11,13) + \sum d(2,12,15)$</td>
<td>$AD + AB'C + A'BD'$</td>
</tr>
<tr>
<td>3 $Y (A, B, C, D) = \sum (0, 5, 6, 7, 9, 10, 13, 14, 15)$</td>
<td>$A'B'C'D' + AC'D + ACD' + BD + BC$</td>
</tr>
<tr>
<td>4 $Y (A,B,C,D) = \sum m(0,3,5,6,7,10,12,13) + \sum d(2,9,15)$</td>
<td>$W'X'Z' + W'Y + X'YZ' + XZ + WXY'$</td>
</tr>
<tr>
<td>5 $Y (W,X,Y,Z) = \sum m(4,5,6,8,9,10,13) + \sum d(0,7,15)$</td>
<td>$AB'D' + AC'D + A'B$</td>
</tr>
<tr>
<td>6 $Y (W,X,Y,Z) = \sum m(5,7,9,11,13,15)$</td>
<td>$XZ + WZ$</td>
</tr>
<tr>
<td>7 $Y (W,X,Y,Z) = \sum m(2,3,6,7,8,10,11,12,14,15)$</td>
<td>$WZ' + Y$</td>
</tr>
<tr>
<td>8 $Y (W,X,Y,Z) = \sum m(0,1,2,3,5,7,13,15)$</td>
<td>$A'B' + BD$</td>
</tr>
<tr>
<td>9 $Y (W,X,Y,Z) = \sum m(1,2,4,8)$</td>
<td>$W'X'Y'Z + W'X'YZ' + W'XY'Z' + WX'Y'Z'$</td>
</tr>
</tbody>
</table>

Table 15: Sample results when the Boolean minimizer is used to reduce selected function
CHAPTER 7: SUMMARY, CONCLUSION AND RECOMMENDATIONS

SUMMARY
The objective of the project was to design a Boolean Function Minimizer which can perform minimization for functions with up to 8 variable and possibly its implementation. This was achieved by developing a software that implemented the QM procedure. Once the reduced function has been obtained a netlist is developed for the reduced function which when we input in a suitable circuit software will generate the logic circuit represented by the reduced function.

The software can be categorized into 3 stages:

Input Stage
Here there are two scenarios: For the case without don’t care and case with don’t care.
For the case without don’t care the values of number of variables, number of minterms, letters to represent literals and the individual minterms. Validation checks are performed at every stage to ensure correct data is entered.
For the case without don’t care the values of number of variables, number of minterms, number of don’t care minterms, letters to represent literals, individual minterms and the don’t care minterms. Validation checks are performed at every stage to ensure correct data is entered.

Minimization Stage
Here the minterms entered are grouped according the number of 1s in their binary equivalent and then paired. The essential prime implicants are then determined and the final reduced function is printed.

Final Stage:
This is where the netlist of the reduced function. When this netlist is entered in a suitable software it generates the logic circuit.

C++ programming language was used in developing the software due to a combination of factors. The most important is that C++ compilers and development kits are easily available. It also supports modular programming and object oriented concepts.

CONCLUSIONS
A Boolean function minimizer software was designed and implemented. The software developed was able to perform Boolean function minimization for up to 8 variables and implement the reduced function.

The project was done with minimal resources and the resulting software had a simple design but was very efficient.

Therefore, the objective of the project was achieved.

RECOMMENDATIONS
With the available time and resources, the objective of the project was met. The project is able to be implemented on a much larger scale. The software can be modified to perform minimization for Boolean Functions with more than 8 variables.

Also, it is possible with a few modifications to integrate this software with a suitable simulation software so as to generate the circuit from the logic functions. One only needs to enter the minterms of the logic function and the logic circuit to be generated.

The software can also be modified to accept all manner of inputs be it minterms, maxterms and actual functions.
Bibliography


# APPENDIXES

## Appendix One: Code used in Welcome screen

```cpp
#pragma once
#include "FormA.h"
#include "FormB.h"

namespace minWdc2 {

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

/// <summary>
/// Summary for Form1
/// </summary>
public ref class Form1 : public System::Windows::Forms::Form
{
    public:
        Form1(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~Form1()
        {
            if (components)
            {
                delete components;
            }
        }

    private: System::Windows::Forms::Label^ label1;
    private: System::Windows::Forms::Button^ button1;
    private: System::Windows::Forms::Button^ button2;
    private: System::Windows::Forms::Button^ button3;
    private: System::Windows::Forms::Label^ label2;

    protected:
        /// <summary>
        /// Required designer variable.
        /// </summary>
        System::ComponentModel::Container ^components;

    #pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->button1 = (gcnew System::Windows::Forms::Button());
        this->button2 = (gcnew System::Windows::Forms::Button());
        this->button3 = (gcnew System::Windows::Forms::Button());
    }
```
```csharp
this->label2 = (gcnew System::Windows::Forms::Label());
this->SuspendLayout();
//
// label1
//
this->label1->AutoSize = true;
this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12, System::Drawing::.FontStyle::Bold, System::Drawing::GraphicsUnit::Point, static_cast<System::Byte>(0)));
this->label1->Location = System::Drawing::Point(42, 28);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(432, 25);
this->label1->TabIndex = 0;
this->label1->Text = L"Welcome to my Boolean Function Minimizer";
//
// button1
//
this->button1->Location = System::Drawing::Point(102, 131);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(121, 87);
this->button1->TabIndex = 1;
this->button1->Text = L"Without Don't care";
this->button1->UseVisualStyleBackColor = true;
this->button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
//
// button2
//
this->button2->Location = System::Drawing::Point(304, 131);
this->button2->Name = L"button2";
this->button2->Size = System::Drawing::Size(121, 87);
this->button2->TabIndex = 2;
this->button2->Text = L"With Don't care";
this->button2->UseVisualStyleBackColor = true;
this->button2->Click += gcnew System::EventHandler(this, &Form1::button2_Click);
//
// button3
//
this->button3->Location = System::Drawing::Point(191, 254);
this->button3->Name = L"button3";
this->button3->Size = System::Drawing::Size(140, 49);
this->button3->TabIndex = 3;
this->button3->Text = L"Exit";
this->button3->UseVisualStyleBackColor = true;
this->button3->Click += gcnew System::EventHandler(this, &Form1::button3_Click);
//
// label2
//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(99, 79);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(326, 17);
this->label2->TabIndex = 4;
this->label2->Text = L"Select an option for Boolean Function Minimization";
//
// Form1
//
this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(521, 362);
this->Controls->Add(this->label2);
```
this->Controls->Add(this->button3);
this->Controls->Add(this->button2);
this->Controls->Add(this->button1);
this->Controls->Add(this->label1);
this->MaximizeBox = false;
this->Name = L"Form1";
this->ShowIcon = false;
this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;
this->Text = L"minWdc";
this->Load += gcnew System::EventHandler(this, &Form1::Form1_Load);
this->ResumeLayout(false);
this->PerformLayout();
}
#pragma endregion
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
}
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
this->Close();
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
Form^ frmA = gcnew FormA(this);
frmA->Show();
this->Hide();
}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
Form^ frmB = gcnew FormB(this);
frmB->Show();
this->Hide();
}

Appendix Three: Code used in the form for minimization for cases where there are ‘don’t care minterms’.
#pragma once
#include <string>
#include <vector>
#include <time.h>
#include "FormA1.h"
#include "FormA2.h"
namespace minWdc2 {
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::IO;

/// <summary>
/// Summary for FormA
/// </summary>
public ref class FormA : public System::Windows::Forms::Form
{
public:
    FormA(Form^ frm1)
    {
        InitializeComponent();
    //
TODO: Add the constructor code here
//
frmMain = Frm1;

protected:
/// <summary>
/// Clean up any resources being used.
/// </summary>
FormA()
{
if (components)
{
    delete components;
}
}

private: char lit;
private: System::Windows::Forms::TextBox^ textBox1;
private: System::Windows::Forms::Label^ label3;
private: System::Windows::Forms::Button^ button3;
private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;
private: System::Windows::Forms::TextBox^ textBox2;
private: System::Windows::Forms::TextBox^ textBox3;
private: System::Windows::Forms::Form^ frmMain;

public: int minWdc1(const int n_var, const int n_min, int total_min, char **lit, int **MIN)
{
    int smin, unmin_limit;
    int count = 0;
    //Conversion to Binary Form
    int **Dec = new int *[n_min];
    for (count = 0; count <= n_min; count++)
    {
        *(Dec + count) = new (std::nothrow) int[n_var];
    }
    int tz,s,y,z,p1;
    for(s=0;scn_min;s++)
    {
        tz=MIN[s][0];
        for(y=n_var;y>0;y--)
        {
            z=y-1;
            p1=tz%2;
            Dec[s][z]=p1;
            tz=tz/2;
        }
    }
    //sorting depending on number of 1s
    int j,k,sum;
    for(j=0;j<n_min;j++)
    {
        sum=0;
        for(k=0;k<n_var;k++)
        {
            if(Dec[j][k]==1)
            {
                sum=sum+1;
            }
        }
        MIN[j][1]=sum;
    }
}
```c
//int Table1[n_min][2];
int **Table1 = new int *[n_min];
for (count = 0; count <= n_min; count++)
{
    *(Table1 + count) = new (std::nothrow) int[2];
}

//int Freq[n_var+1][2];
int **Freq = new int *[n_var+1];
for (count = 0; count <= (n_var+1); count++)
{
    *(Freq + count) = new (std::nothrow) int[2];
}

int n_check,n_t1,l1,m_freq,n_freq,t04,t02,t03;
int t01 = 0;
n_check=0;
m_freq=0;

recheck:
    n_freq=0;
for(l1=0;l1<n_min;l1+)
{
    if(MIN[l1][1]==n_check)
    {
        Table1[n_t1][0]=MIN[l1][0];
        n_t1=n_t1+1;
        n_freq=n_freq+1;
    }
}
if(n_freq>0)
    Freq[m_freq][0]=n_freq;
++m_freq;
if(n_check<n_var||n_check==n_var)
    n_check=n_check+1;
goto recheck;

if(m_freq>0)
    Freq[0][1]=Freq[0][0];
}
if(m_freq>1)
{
    for(t02=1;t02<m_freq;t02++)
    {
        t04=t02-1;
        t03=Freq[t04][1]+ Freq[t02][0];
        Freq[t02][1]=t03;
    }
}
//Matching
int m_pair,count_groups,tst,m1,m2,m3,t,pair_each,x,t1,y1,t2,diff;
int pzt,N_PI,t10,t11,t12,t13,t14,t15,P_freq;
P_freq=0;
//int MID[256][256];
int **MID = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(MID + count) = new (std::nothrow) int[256];
}
//int FINAL[256][256];
int **FINAL = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(FINAL + count) = new (std::nothrow) int[256];
}
```

//int FINAL_FREQ[256][2];
int **FINAL_FREQ = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(FINAL_FREQ + count) = new (std::nothrow) int[2];
}

//int FINAL_diff[256];
int **FINAL_diff = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(FINAL_diff + count) = new (std::nothrow) int[256];
}

//int Table_diff[256][256];
int **Table_diff = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(Table_diff + count) = new (std::nothrow) int[256];
}

//int Mid_freq[256][256];
int **Mid_freq = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(Mid_freq + count) = new (std::nothrow) int[256];
}

//int CHECK[256][2];
int **CHECK = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(CHECK + count) = new (std::nothrow) int[2];
}

//int PI[256][256];
int **PI = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(PI + count) = new (std::nothrow) int[256];
}

//int PI_Ndiff[256];
int *PI_Ndiff = new int[256];

//int PI_diff[256][256];
int **PI_diff = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(PI_diff + count) = new (std::nothrow) int[256];
}

//int PI_Freq[256][2];
int **PI_Freq = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(PI_Freq + count) = new (std::nothrow) int[2];
}

N_PI = 0;
m_pair = 0;
count_groups = 0;
//TODO: Freq is [n_var+1][2] so does nvar have to be > 2, 3 actually
if(m_freq==1){
for(t=0;t<n_min;t++){  
    PI[N_PI][0]=Table1[t][0];  
    PI_Ndiff[N_PI]=0;  
    N_PI=N_PI+1;  
}  
goto Fastfoward;  
}  
if(m_freq>1){  
    for(t=0;t<m_freq-1;t++){  
        tst=Freq[t][1];  
        m1=tst-Freq[t][0];  
        m2=Freq[t+1][1];  
        m3=m2-Freq[t+1][0];  
        pair_each=0;  
        for(x=m1;x<tst;x++){  
            t1=Table1[x][0];  
            for(y1=m3;y1<m2;y1++){  
                t2=Table1[y1][0];  
                diff=t2-t1;  
                if(diff==1||diff==2||diff==4||diff==8||diff==16||diff==32||diff==64||diff==128){  
                    MID[m_pair][0]=t1;  
                    MID[m_pair][1]=t2;  
                    Table_diff[m_pair][0]=diff;  
                    m_pair=m_pair+1;  
                    pair_each=pair_each+1;  
                    Table1[x][1]=1;  
                    Table1[y1][1]=1;  
                }  
            }  
        }  
        if(pair_each>0){  
            Mid_freq[count_groups][0]=pair_each;  
            count_groups=count_groups+1;  
        }  
    }  
}  
for (pzt=0;pzt<n_min;pzt++){  
    if(Table1[pzt][1]!=1){  
        PI[N_PI][0]=pzt;  
        PI_Ndiff[N_PI]=0;  
        N_PI=N_PI+1;  
        P_freq=P_freq+1;  
    }  
}  
if (count_groups==1){  
    for(t=0;t<m_pair;t++){  
        PI[N_PI][0]=MID[t][0];  
        PI[N_PI][1]=MID[t][1];  
        PI_Ndiff[N_PI]=1;  
        N_PI=N_PI+1;  
    }  
}  
goto Fastfoward;  
}

int tst1,tst2,tst3,tst4,tst5,i1,m11,m21,m31,x1,t31,t21,t41,t51,y2,diff_1;  
int new_pair,total_pair,total_pair1;  
int j3,j4;  
if(count_groups>1){  
    Mid_freq[0][1]=Mid_freq[0][0];  
    for(t12=1;t12<count_groups;t12++){
t14=t12-1;
t13=Mid_freq[t14][1]+ Mid_freq[t12][0];
Mid_freq[t12][1]=t13;
}

tst1=1;
m11=count_groups-1;
tst4=tst1*2;
tst5=tst4*2;
count_groups=0;
total_pair=0;
for(i1=0;i1<m11;i1++){
  new_pair=0;
tst2=Mid_freq[i1][1];
m21=Mid_freq[i1][1]-Mid_freq[i1][0];
tst3=Mid_freq[i1+1][1];
m31=Mid_freq[i1+1][1]-Mid_freq[i1+1][0];
  for(x1=m21;x1<tst2;x1++){
    t21=Table_diff[x1][0];
    for(y2=m31;y2<tst3;y2++){
      t31=Table_diff[y2][0];
      if(t21==t31){
        diff_1=MID[y2][0]-MID[x1][0];
        if(diff_1==1||diff_1==2||diff_1==4||diff_1==8||diff_1==16||diff_1==32||diff_1==64||diff_1==128)
          FINAL[total_pair][0]=MID[x1][0];FINAL[total_pair][1]=MID[x1][1];FINAL[total_pair][2]=
          MID[y2][0];FINAL[total_pair][3]=MID[y2][1];
          FINAL_diff[total_pair][0]=t21;
          FINAL_diff[total_pair][1]=diff_1;
          CHECK[x1][0]=1;
          CHECK[y2][0]=1;
          new_pair=new_pair+1;
          total_pair=total_pair+1;
      }
    }
  }
}

if(new_pair>0){
  FINAL_FREQ[count_groups][0]=new_pair;
  count_groups=count_groups+1;
}

if(total_pair>0){
total_pair1=total_pair;
}

for(j4=0;j4<m_pair;j4++){
  if(CHECK[j4][0]!=1){
    PI[N_PI][0]=MID[j4][0];
    PI[N_PI][1]=MID[j4][1];
    PI_Ndiff[N_PI]=1;
    N_PI=N_PI+1;
    P_freq=P_freq+1;
  }
}

int j5;
if(count_groups==1){
  for(j5=0;j5<total_pair;j5++){
PI[N_PI][0]=FINAL[j5][0];
PI[N_PI][1]=FINAL[j5][1];
PI[N_PI][2]=FINAL[j5][2];
PI[N_PI][3]=FINAL[j5][3];
PI_Ndiff[N_PI]=2;
N_PI=N_PI+1;
P_freq=P_freq+1;
}
goto Fastfoward;
}
if(count_groups>1){
//sort function
for (j3=0;j3<total_pair;j3++){
    if(FINAL_diff[j3][0]>FINAL_diff[j3][1]){
        j4=FINAL_diff[j3][0];
        FINAL_diff[j3][0]=FINAL_diff[j3][1];
        FINAL_diff[j3][1]=j4;
    }
}
FINAL_FREQ[0][1]=FINAL_FREQ[0][0];
t15=FINAL_FREQ[0][1];
for(t12=1;t12<count_groups;t12++){
    t14=t12-1;
    t13=FINAL_FREQ[t14][1]+ FINAL_FREQ[t12][0];
    FINAL_FREQ[t12][1]=t13;
}
tst1=2;
m11=count_groups-1;
tst4=tst1*2;
tst5=tst4*2;
count_groups=0;
total_pair=0;
for(i1=0;i1<m11;i1++){
    new_pair=0;
    tst2=FINAL_FREQ[i1][1];
m21=FINAL_FREQ[i1][1]-FINAL_FREQ[i1][0];
m31=FINAL_FREQ[i1+1][1]-FINAL_FREQ[i1+1][0];
    for(x1=m21;x1<tst2;x1++){
        t21=FINAL_diff[x1][0];
        t31=FINAL_diff[x1][1];
        for(y2=m31;y2<tst3;y2++){
            t41=FINAL_diff[y2][0];
            t51=FINAL_diff[y2][1];
            if(t21==t41){
                if(t31==t51){
                    diff_1=FINAL[y2][0]-FINAL[x1][0];
                    if(diff_1==1||diff_1==2||diff_1==4||diff_1==8||diff_1==16||diff_1==32||diff_1==64||diff_1==128){
                        MID[total_pair][0]=FINAL[x1][0];MID[total_pair][1]=FINAL[x1][1];MID[total_pair][2]=FINAL[x1][2];MID[total_pair][3]=FINAL[x1][3];
                        MID[total_pair][4]=FINAL[y2][0];MID[total_pair][5]=FINAL[y2][1];MID[total_pair][6]=FINAL[y2][2];MID[total_pair][7]=FINAL[y2][3];
                    }
                }
            }
        }
    }
}
if(diff_1==1||diff_1==2||diff_1==4||diff_1==8||diff_1==16||diff_1==32||diff_1==64||diff_1==128){
    MID[total_pair][0]=FINAL[x1][0];MID[total_pair][1]=FINAL[x1][1];MID[total_pair][2]=FINAL[x1][2];MID[total_pair][3]=FINAL[x1][3];
    MID[total_pair][4]=FINAL[y2][0];MID[total_pair][5]=FINAL[y2][1];MID[total_pair][6]=FINAL[y2][2];MID[total_pair][7]=FINAL[y2][3];
}
if (new_pair > 0) {
    FINAL_FREQ[count_groups][0] = new_pair;
    count_groups = count_groups + 1;
}

if (total_pair > 0) {
    total_pair1 = total_pair;
}
for (j4 = 0; j4 < total_pair1; j4++) {
    if (CHECK[j4][1] != 1) {
        PI[N_PI][0] = FINAL[j4][0];
        PI[N_PI][1] = FINAL[j4][1];
        PI[N_PI][2] = FINAL[j4][2];
        PI[N_PI][3] = FINAL[j4][3];
        PI_Ndiff[N_PI] = 2;
        N_PI = N_PI + 1;
        P_freq = P_freq + 1;
    }
}
if (count_groups == 1) {
    for (j5 = 0; j5 < total_pair; j5++) {
        PI[N_PI][0] = MID[j5][0];
        PI[N_PI][1] = MID[j5][1];
        PI[N_PI][2] = MID[j5][2];
        PI[N_PI][3] = MID[j5][3];
        PI[N_PI][4] = MID[j5][4];
        PI[N_PI][5] = MID[j5][5];
        PI[N_PI][6] = MID[j5][6];
        PI[N_PI][7] = MID[j5][7];
        PI_Ndiff[N_PI] = 3;
        N_PI = N_PI + 1;
        P_freq = P_freq + 1;
    }
    goto Fastfoward;
}
// sorting
Fastfoward:
int xy, tsp, tsp1, b, temp;
// int COVER_NTERMS[9];
int *COVER_NTERMS = new int[9];

for (xy = 0; xy < N_PI; xy++) {
    rudia:
    tsp = PI_Ndiff[xy];
    tsp1 = COVER_NTERMS[tsp];
    for (j = 0; j < tsp1 - 1; j++) {
        if (PI[xy][j] > PI[xy][j + 1]) {
            temp = PI[xy][j];
            PI[xy][j] = PI[xy][j + 1];
            PI[xy][j + 1] = temp;
        }
    }
}
PI[xy][j+1]=temp;
}

b=b+1;
if(b<tsp1){
    goto rudia;
}

//int Dupli_num[N_PI];
int *Dupli_num = new int [N_PI];
//int PI1 [N_PI][256];
int **PI1 = new int *[N_PI];
for (count = 0; count <=(N_PI); count++)
{
    *(PI1 + count) = new (std::nothrow) int[256];
}
int M_PI,yx;
M_PI=0;
tsp=PI_Ndiff[0];
tsp1=COVER_NTERMS[tsp];
int id,jd,kd,ld,md,d_copy,tps,tps1;
for(id=0;id<N_PI;id++)
{
    d_copy=0;
    jd=id+1;
    for(kd=0;kd<4;kd++)
    {
        if(PI[id][kd]==PI[jd][kd])
        {
            d_copy=d_copy+1;
        }
    }
    if(d_copy==4){
        for(ld=jd;ld<N_PI;ld++)
        {
            tps=ld-1;
            PI[tps][0]=PI[id][0];
            PI[tps][1]=PI[id][1];
            PI[tps][2]=PI[id][2];
            PI[tps][3]=PI[id][3];
            PI_Ndiff[tps]=PI_Ndiff[ld];
        }
    }
}
for(yx=0;yx<N_PI;yx++)
{
    Dupli_num[yx]=PI[yx][0]+PI[yx][1]+PI[yx][2]+PI[yx][3];
}
int sum_dupli;
sum_dupli=1;
for(xy=0;xy<N_PI-1;xy++)
{
    if(Dupli_num[xy]!=Dupli_num[xy+1]){
        sum_dupli=sum_dupli+1;
    }
    else{
        sum_dupli=sum_dupli+0;
    }
}
N_PI=sum_dupli;
//Developing cover chart
//int COVER_CHART[N_PI+1][n_min+1];
int **COVER_CHART = new int *[N_PI+1];
for (count = 0; count <=(N_PI+1); count++)
{
}
*(COVER_CHART + count) = new (std::nothrow) int[n_min+1];
}
int j6,j7,j8,tpz,tpz1,j9;
//int COVER_CHECK[n_min];
int *COVER_CHECK = new int [n_min];

for(j6=0;j6<N_PI;j6++){
    for(j7=0;j7<n_min;j7++){
        COVER_CHART[j6][j7]=0;
    }
}
//int NOT[N_PI];
int *NOT = new int [N_PI];

j7=0;
for(j6=0;j6<N_PI;j6++){
    tps=PI_Ndiff[j6];
    tps1=COVER_NTERMS[tps];
    NOT[j7]=tps1;
    j7=j7+1;
}
int tps2,tps3,tps4;
for(j6=0;j6<N_PI;j6++){
    tps=NOT[j6];
    for(tps1=0;tps1<tps;tps1++){
        tps2=PI[j6][tps1];
        for(tps3=0;tps3<n_min;tps3++){
            tps4=MIN[tps3][0];
            if(tps2==tps4){
                COVER_CHART[j6][tps3]=1;
            }
        }
    }
}

int N_sum;
for(j7=0;j7<n_min;j7++){
    N_sum=0;
    for(j6=0;j6<N_PI;j6++){
        N_sum=N_sum+COVER_CHART[j6][j7];
    }
    COVER_CHART[N_PI][j7]=N_sum;
}
//initialize min check
for(j7=0;j7<n_min;j7++){
    COVER_CHECK[j7]=0;
}

//check each individual PI to determine essential pi
//int EPI_check[N_PI];
int *EPI_check = new int [N_PI];

for(j7=0;j7<N_PI;j7++){
    EPI_check[j7]=0;
}
for(j7=0;j7<N_PI;j7++){
    EPI_check[j7]=0;
}
for(tps=0;tps<n_min;tps++){
    tps1=COVER_CHART[N_PI][tps];
    if(tps1=1){
        for (j6=0;j6<N_PI;j6++){
            
        }
    }
}
if(COVER_CHART[j6][tps]==1){
    EPI_check[j6]=1;
}
}
}

//test if all minterms are covered
for(j9=0;j9<N_PI;j9++){
    if(EPI_check[j9]==1){
        j6=NOT[j9];
        for(j7=0;j7<j6;j7++){
            tps=PI[j9][j7];
            for(j8=0;j8<n_min;j8++){
                if(tps==MIN[j8][0]){  
                    COVER_CHECK[j8]=1;
                }
            }
        }
    }
}

int N_sum1;
N_sum1=0;
for(j6=0;j6<n_min;j6++){
    if(COVER_CHECK[j6]==1){
        N_sum1=N_sum1+1;
    }
}

int epi;
epi=0;

//int EPI[N_PI][n_var];
int **EPI = new int *[N_PI];
for (count = 0; count <=(N_PI); count++)
{
    *(EPI + count) = new (std::nothrow) int[n_var];
}

if(N_sum1==n_min){
    for(j6=0;j6<N_PI;j6++){
        if(EPI_check[j6]==1){
            tps=PI[j6][0];
            for(j7=n_var;j7>0;j7--){
                tps1=j7-1;
                tps2=tps%2;
                EPI[epi][tps1]=tps2;
                tps=tps/2;
            }
            epi=epi+1;
        }
    }
}

//INSERTING THE DONT CARE CONDITIONS
int p_dcare,j80;
j80=0;

if(N_sum1==n_min){
    for(j6=0;j6<N_PI;j6++){
        if(EPI_check[j6]==1){
            tps=PI_Ndiff[j6];
            if(tps>0){
                tps1=2;
            }
        }
    }
}
tps2=tps1-1;
for(j7=0; j7<tps; j7++){
    p_dcare=PI[j6][tps2]-PI[j6][0];
    tps2=tps2*2;
    tps3=n_var-1;
    tps4=1;
    for(j8=0; j8<n_var; j8++){
        if(p_dcare==tps4){
            EPI[j80][tps3]=2;
        }
        tps3=tps3-1;
        tps4=tps4*2;
    }
}
}
}
}
}
}
}

//CASES WHERE THE ESSENTIAL PIs do not cover all the minterms
int uncovered, uc, uc_sum;
uc=0;
if(N_sum1!=n_min){
    uncovered=n_min-N_sum1;
    //int UNC[N_PI][uncovered+1];
    int **UNC = new int *[N_PI];
    for (count = 0; count <(N_PI); count++){
        *(UNC + count) = new (std::nothrow) int[uncovered+1];
    }
}
for(j6=0; j6<n_min; j6++){  
    if(COVER_CHECK[j6]==0){
        for(j7=0; j7<N_PI; j7++){  
            UNC[j7][uc]=COVER_CHART[j7][j6];
        }
        uc=uc+1;
    }
}

//Determining other essential prime implicants
//First develop a table for the unchecked minterms
//int Table[uncovered][3];
int **Table = new int *[uncovered];
for (count = 0; count <=(uncovered); count++){
    *(Table + count) = new (std::nothrow) int[3];
}
int uc2;
for(j6=0; j6<n_min; j6++){  
    if(COVER_CHECK[j6]==0){
        Table[uc2][0]=MIN[j6][0];
        Table[uc2][1]=j6;
        uc2=uc2+1;
    }
}

recheck_pi:
//find sum of columns
for(j6=0; j6<N_PI; j6++){
uc_sum=0;
for(j7=0;j7<uc;j7++){
    uc_sum=uc_sum+UNC[j6][j7];
}
UNC[j6][uncovered]=uc_sum;

//Find largest sum in the UNC LAST COLUMN
int t_largest,t1;
t_largest=UNC[0][uncovered];
t1=0;
for(j6=1;j6<N_PI;j6++){
    if(t_largest<UNC[j6][uncovered]){
        t_largest=UNC[j6][uncovered];
        t1=j6;
    }
}
EPI_check[t1]=1;
tps=NOT[t_largest];
for(j7=0;j7<tps;j7++){
    tps1=PI[t1][j7];
    for(j8=0;j8<n_min;j8++){
        if(tps1==MIN[j8][0]){
            COVER_CHECK[j8]=1;
        }
    }
}

int N_sum2;
N_sum2=COVER_CHECK[0];
for(j6=1;j6<n_min;j6++){
    N_sum2=N_sum2+COVER_CHECK[j6];
}
if(N_sum2==n_min){
    for(j6=0;j6<N_PI;j6++){
        if(EPI_check[j6]==1){
            tps=PI[j6][0];
            for(j7=n_var;j7>0;j7--){
                tps1=j7-1;
                tps2=tps%2;
                EPI[epi][tps1]=tps2;
                tps=tps/2;
            }
            epi=epi+1;
        }
    }
}
for(j6=0;j6<N_PI;j6++){
    if(EPI_check[j6]==1){
        tps=PI_Ndiff[j6];
        if(tps>0){
            tps1=2;
            tps2=tps1-1;
            for(j7=0;j7<tps;j7++){
                p_dcare=PI[j6][tps2]-PI[j6][0];
                tps2=tps2*2;
                tps3=n_var-1;
                tps4=1;
                for(j8=0;j8<n_var;j8++){
                    if(p_dcare==tps4){
                        EPI[j8][tps3]=2;
                    }
                    tps3=tps3-1;
                    tps4=tps4*2;
                }
            }
        }
    }
}
if(N_sum2!=n_min){
  tps=NOT[t1];
  //TODO: Is this necessary?
  //cout<<tps<<endl;
  for(j6=0;j6<tps;j6++){
    tps1=PI[t1][j6];
    for(j7=0;j7<uncovered;j7++){
      if(tps1==Table[j7][0]){
        for(j8=0;j8<N_PI;j8++){
          UNC[j8][j7]=0;
        }
      }
    }
  }
  goto recheck_pi;
}

//DISPLAYING THE MINIMIZED FUNCTION
//cout<<"THE FINAL REDUCED FUNCTION IS\n";
for(j6=0;j6<epi;j6++){
  for(j7=0;j7<n_var;j7++){
    if(EPI[j6][j7]==1){
      this->textBox5->Text += gcnew String(lit[j7]);
    }
  }
  if(j6<epi-1){
    this->textBox5->Text += " + ";
  }
}

//IMPLEMENTATION(GENERATING NETLIST)
//String Newline = Environment.NewLine;
this->textBox1->Text = "THE IMPLEMENTATION OF THE REDUCED FUNCTION IS GIVEN BY THE NETLIST BELOW\n";
this->textBox1->Text += "\n"
this->textBox1->Text += "IS SCHEMATIC DESCRIPTION FORMAT 8.0\n";
this->textBox1->Text += "Design: zade.pdsprj\n"
char date[9];
_strdate(date);
this->textBox1->Text += gcnew System::String(date);
this->textBox1->Text += "\n";
76
this->textBox1->Text += "Modified: ";
_strdate(date);
this->textBox1->Text += gcnew System::String(date) + "\r\n";
this->textBox1->Text += "PROPERTIES,0\r\n";
this->textBox1->Text += "MODELDEFS,0\r\n";
this->textBox1->Text += "PARTLIST,2\r\n";
this->textBox1->Text += "NETLIST,7\r\n";
//int IMP_TABLE[epi];
int *IMP_TABLE = new int [epi];
int IMP;
IMP=0;
for(j=0;j<epi;j++){
    for(k=0;k<n_var;k++){
        if(EPI[j][k]==0|EPI[j][k]==1){
            IMP=IMP+1;
        }
    }
    IMP_TABLE[j]=IMP;
    IMP=0;
}
int T_IMP,T_IMP1,T_IMP2,T_IMP3,j_count;
j_count=0;
T_IMP1=0;
T_IMP2=IMP_TABLE[0]+1;
int j1;
for(j=0;j<epi;j++){
    this->textBox1->Text += ",2\r\n";
    T_IMP=IMP_TABLE[j];
    for(k=0;k<T_IMP;k++){
        T_IMP1=T_IMP1+1;
        this->textBox1->Text += "U1";this->textBox1->Text += ",IP,";this->textBox1->Text += T_IMP1;
        this->textBox1->Text += "\r\n";
    }j1=j1+1;
    this->textBox1->Text += ",2\r\n";
    this->textBox1->Text += "#0000";this->textBox1->Text += ",OP,";this->textBox1->Text += T_IMP2;
    this->textBox1->Text += "\r\n";
    this->textBox1->Text += "U2";this->textBox1->Text += ",IP,";this->textBox1->Text += "D";this->textBox1->Text += j_count;
    this->textBox1->Text += "\r\n";
    j_count= j_count +1;
    if (j==0){
        T_IMP1=T_IMP+epi;
    }
j1=j1+1;
    T_IMP2=T_IMP2+1;
this->textBox1->Text += "\r\n";
}

//IMPLEMENTATION

77
```csharp
private: System::Windows::Forms::Label^ label1;
protected:
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::TextBox^ textBox5;
private: System::Windows::Forms::Label^ label5;
private: System::Windows::Forms::Button^ button1;
private: System::Windows::Forms::Button^ button2;

// <summary>
// Required designer variable.
// </summary>
System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
void InitializeComponent(void)
{
    this->label1 = (gcnew System::Windows::Forms::Label());
    this->label2 = (gcnew System::Windows::Forms::Label());
    this->textBox5 = (gcnew System::Windows::Forms::TextBox());
    this->label5 = (gcnew System::Windows::Forms::Label());
    this->button1 = (gcnew System::Windows::Forms::Button());
    this->button2 = (gcnew System::Windows::Forms::Button());
    this->textBox1 = (gcnew System::Windows::Forms::TextBox());
    this->label3 = (gcnew System::Windows::Forms::Label());
    this->button3 = (gcnew System::Windows::Forms::Button());
    this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());
    this->textBox2 = (gcnew System::Windows::Forms::TextBox());
    this->textBox3 = (gcnew System::Windows::Forms::TextBox());
    this->SuspendLayout();
    // label1
    //
    this->label1->AutoSize = true;
    this->label1->Location = System::Drawing::Point(45, 29);
    this->label1->Name = L"label1";
    this->label1->Size = System::Drawing::Size(137, 17);
    this->label1->TabIndex = 0;
    this->label1->Text = L"Number of Variables";
    //
    // label2
    //
    this->label2->AutoSize = true;
    this->label2->Location = System::Drawing::Point(45, 69);
    this->label2->Name = L"label2";
    this->label2->Size = System::Drawing::Size(135, 17);
    this->label2->TabIndex = 2;
    this->label2->Text = L"Number of Minterms";
}

return 0;
```

// textBox5
//
this->textBox5->Enabled = false;
this->textBox5->Location = System::Drawing::Point(239, 216);
this->textBox5->Name = L"textBox5";
this->textBox5->Size = System::Drawing::Size(295, 22);
this->textBox5->TabIndex = 9;

// label5
//
this->label5->AutoSize = true;
this->label5->Location = System::Drawing::Point(45, 216);
this->label5->Name = L"label5";
this->label5->Size = System::Drawing::Size(176, 17);
this->label5->TabIndex = 8;
this->label5->Text = L"Obtained reduced function";

// button1
//
this->button1->Location = System::Drawing::Point(239, 113);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(156, 52);
this->button1->TabIndex = 11;
this->button1->Text = L"Next";
this->button1->UseVisualStyleBackColor = true;
FormA::button1_Click();

// button2
//
this->button2->Location = System::Drawing::Point(48, 459);
this->button2->Name = L"button2";
this->button2->Size = System::Drawing::Size(82, 30);
this->button2->TabIndex = 12;
this->button2->Text = L"<< Back";
this->button2->UseVisualStyleBackColor = true;
FormA::button2_Click();

// textBox1
//
this->textBox1->AcceptsReturn = true;
this->textBox1->BackColor = System::Drawing::SystemColors::HighlightText;
this->textBox1->Location = System::Drawing::Point(48, 287);
this->textBox1->Name = L"textBox1";
this->textBox1->Multiline = true;
this->textBox1->ReadOnly = true;
this->textBox1->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;
this->textBox1->Size = System::Drawing::Size(486, 166);
this->textBox1->TabIndex = 14;

// label3
//
this->label3->AutoSize = true;
this->label3->Location = System::Drawing::Point(45, 267);
this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(65, 17);
this->label3->TabIndex = 15;
this->label3->Text = L"NETLIST";

// button3
//
// this->button3->Location = System::Drawing::Point(436, 455);
// this->button3->Name = L"button3";
// this->button3->Size = System::Drawing::Size(98, 34);
// this->button3->TabIndex = 28;
// this->button3->Text = L"Save";
// this->button3->UseVisualStyleBackColor = true;
// this->button3->Click += gcnew System::EventHandler(this,
// &FormA::button3_Click);

//
// saveFileDialog1
// this->saveFileDialog1->Filter = L"Text File|*.txt";
// this->saveFileDialog1->RestoreDirectory = true;
//
// textBox2
// this->textBox2->Location = System::Drawing::Point(239, 29);
// this->textBox2->Name = L"textBox2";
// this->textBox2->Size = System::Drawing::Size(125, 22);
// this->textBox2->TabIndex = 29;
// this->textBox2->TextChanged += gcnew System::EventHandler(
// this,
// &FormA::textBox2_TextChanged_1);

//
// textBox3
// this->textBox3->Location = System::Drawing::Point(239, 69);
// this->textBox3->Name = L"textBox3";
// this->textBox3->Size = System::Drawing::Size(125, 22);
// this->textBox3->TabIndex = 30;
// this->textBox3->TextChanged += gcnew System::EventHandler(
// this,
// &FormA::textBox3_TextChanged);

//
// FormA
// this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
// this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
// this->ClientSize = System::Drawing::Size(575, 504);
// this->Controls->Add(this->textBox3);
// this->Controls->Add(this->textBox2);
// this->Controls->Add(this->button3);
// this->Controls->Add(this->label13);
// this->Controls->Add(this->textBox1);
// this->Controls->Add(this->button2);
// this->Controls->Add(this->button1);
// this->Controls->Add(this->textBox5);
// this->Controls->Add(this->label5);
// this->Controls->Add(this->label2);
// this->Controls->Add(this->label1);
// this->MaximizeBox = false;
// this->Name = L"FormA";
// this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;
// this->Text = L"Without Don’t care";
// this->ResumeLayout(false);
// this->PerformLayout();

#pragma endregion
private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) { 
   frmMain->Show();
   this->Hide();
}
```csharp
private: System::Void textBox2_TextChanged(System::Object^ sender, System::EventArgs^ e) {
    MessageBox::Show("The", "Error", MessageBoxButtons::OK);
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    //const int n_var = 10;
    int count = 0;
    int n, i, total_min, smin, umin_limit;
    const int n_var = Convert::ToInt32((this->textBox2->Text));
    total_min = 1;
    for (smin = 0; smin < n_var; smin++)
        total_min = 2 * total_min;
    const int n_min = Convert::ToInt32((this->textBox3->Text));
    if (n_min < 1 || n_min > total_min)
        MessageBox::Show("The number of minterms should be between 1 and " + total_min, "Error", MessageBoxButtons::OK);
    return;
}

umin_limit = total_min - 1;
//char lit[n_var][1];
char **lit = new char *[n_var];
for (count = 0; count <= (n_var); count++)
    *(lit + count) = new (std::nothrow) char[1];

for (n = 0; n < n_var; n++)
    FormA1^ frmA1 = gcnew FormA1(n);
    frmA1->ShowDialog();
    //cout << "Enter the letters you want to use to represent the final reduced function starting from MSB\n"
    lit[n][0] = frmA1->getLit();
    lit[n][1] = '\0';

int **MIN = new int *[n_min];
for (count = 0; count <= n_min; count++)
    *(MIN + count) = new (std::nothrow) int[2];

for (i = 0; i < n_min; i++)
    //cout << "Enter Minterm\n"
    FormA2^ frmA2 = gcnew FormA2(i, umin_limit);
    frmA2->ShowDialog();
    MIN[i][0] = frmA2->getMIN();

    this->minWdc1(n_var, n_min, total_min, lit, MIN);
}

private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
    saveFileDialog1->ShowDialog();
    if (saveFileDialog1->FileName != "")
    {
        String^ filename = saveFileDialog1->FileName;
        StreamWriter^ file = gcnew StreamWriter(filename);
        file->Write(textBox1->Text);
        file->Close();
    }
}

private: System::Void textBox2_TextChanged_1(System::Object^ sender, System::EventArgs^ e) {
    if (textBox2->Text->Equals(""))
    {
```
} else if (!Regex::IsMatch(textBox2->Text, "^[0-9]+$")) {
    MessageBox::Show("This textbox accepts only integers", "Error", MessageBoxButtons::OK);
    String^ s = textBox2->Text;
    textBox2->Text = s->Substring(0, s->Length - 1);
    textBox2->SelectAll();
    return;
}

private: System::Void textBox3_TextChanged(System::Object^ sender, System::EventArgs^ e) {
    if (textBox3->Text->Equals(String::Empty)) {
    } else if (!Regex::IsMatch(textBox3->Text, "^[0-9]+$")) {
        MessageBox::Show("This textbox accepts only integers", "Error", MessageBoxButtons::OK);
        String^ s = textBox3->Text;
        textBox3->Text = s->Substring(0, s->Length - 1);
        textBox3->SelectAll();
        return;
    }
}

Appendix Three: Code used in the form for minimization for cases where there are ‘don’t care minterms’
#pragma once
#include <string>
#include <time.h>
#include "FormB1.h"
#include "FormB2.h"
#include "FormB3.h"
namespace minWdc2 {
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::IO;

    /// <summary>
    /// Summary for FormB
    /// </summary>
    public ref class FormB : public System::Windows::Forms::Form
    {
        public:
            FormB(System^ frm1)
            {
                InitializeComponent();
                //
                //TODO: Add the constructor code here
                //
                bol_dc = false;
                checkBox1->Checked = false;
                textBox4->Enabled = false;
                frmMain = frm1;
            }

        protected:
            /// <summary>
            /// Clean up any resources being used.
            /// </summary>

    }
~FormB()
{
    if (components)
    {
        delete components;
    }
}

private: System::Windows::Forms::TextBox^ textBox1;
private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;
private: System::Windows::Forms::Button^ button3;
private: System::Windows::Forms::TextBox^ textBox2;
private: System::Windows::Forms::TextBox^ textBox3;
private: System::Windows::Forms::TextBox^ textBox4;

protected:
private: Boolean bol_dc;
private: int minWdc2(int n_var, int n_min, int n_minOr, int total_min, char **lit, int **MIN){
    //Conversion to Binary Form
    int count =0;
    //int Dec[n_min][n_var];
    int **Dec = new int *[n_min];
    for (count = 0; count <=n_min; count++)
    {
        *(Dec + count) = new (std::nothrow) int[n_var];
    }

    int tz,s,y,z,p1;
    for(s=0;s<n_min;s++){
        tz=MIN[s][0];
        for(y=n_var;y>0;y--){
            z=y-1;
            p1=tz%2;
            Dec[s][z]=p1;
            tz=tz/2;
        }
    }

    //sorting depending on number of 1s
    int j,k,sum;
    for(j=0;j<n_min;j++){
        sum=0;
        for(k=0;k<n_var;k++){
            if(Dec[j][k]==1){
                sum=sum+1;
            }
        }
        MIN[j][1]=sum;
    }
    //int Table1[n_min][2];
    int **Table1 = new int *[n_min];
    for (count = 0; count <=n_min; count++)
    {
        *(Table1 + count) = new (std::nothrow) int[2];
    }
    //int Freq[n_var+1][2];
    int **Freq = new int *[n_var+1];
    for (count = 0; count <=(n_var+1); count++)
    {
        *(Freq + count) = new (std::nothrow) int[2];
    }
    int n_check,n_t1,11,m_freq,n_freq,t04,t02,t03;
n_t1=0;
n_check=0;
m_freq=0;

recheck:

n_freq=0;
for(l1=0;l1<n_min;l1++){
    if(MIN[l1][1]==n_check){
        Table1[n_t1][0]=MIN[l1][0];
        n_t1=n_t1+1;
        n_freq=n_freq+1;
    }
}

if(n_freq){Freq[m_freq][0]=n_freq;
++m_freq;
}

if(n_check<n_var || n_check==n_var){
    n_check=n_check+1;
goto recheck;
}

if(m_freq>0){
    Freq[0][1]=Freq[0][0];
}

if(m_freq>1){
    for(t02=1;t02<m_freq;t02++){
        t04=t02-1;
        t03=Freq[t04][1]+ Freq[t02][0];
        Freq[t02][1]=t03;
    }
}

//Matching
m_pair,count_groups,tst,m1,m2,m3,t,pair_each,x,t1,y1,t2,diff;
int pzt,N_PI,t10,t11,t12,t13,t14,t15,P_freq;
P_freq=0;

//int MID[256][256];
int **MID = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(MID + count) = new (std::nothrow) int[256];
}

//int FINAL[256][256];
int **FINAL = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(FINAL + count) = new (std::nothrow) int[256];
}

//int FINAL_FREQ[256][2];
int **FINAL_FREQ = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(FINAL_FREQ + count) = new (std::nothrow) int[2];
}

//int FINAL_diff[256][256];
int **FINAL_diff = new int *[256];
for (count = 0; count <=(256); count++)
{
    *(FINAL_diff + count) = new (std::nothrow) int[256];
}

//int Table_diff[256][256];
int **Table_diff = new int *[256];
for (count = 0; count <= 256; count++)
    *(Table_diff + count) = new (std::nothrow) int[256];

int **Mid_freq = new int *[256];
for (count = 0; count <= 256; count++)
    *(Mid_freq + count) = new (std::nothrow) int[256];

int **Mid_freq[2];
for (count = 0; count <= 256; count++)
    *(Mid_freq + count) = new (std::nothrow) int[2];

int **PI = new int *[256];
for (count = 0; count <= 256; count++)
    *(PI + count) = new (std::nothrow) int[256];

int **PI[256][2];
for (count = 0; count <= 256; count++)
    *(PI[256][2] + count) = new (std::nothrow) int[256];

int **PI_diff = new int *[256];
for (count = 0; count <= 256; count++)
    *(PI_diff + count) = new (std::nothrow) int[256];

int **PI_Freq = new int *[256];
for (count = 0; count <= 256; count++)
    *(PI_Freq + count) = new (std::nothrow) int[256];

N_PI = 0;
m_pair = 0;
count_groups = 0;
if (m_freq == 1){
    for (t = 0; t < n_min; t++){
        PI[N_PI][0] = Table1[t][0];
        PI_Ndiff[N_PI] = 0;
        N_PI = N_PI + 1;
    }
    goto Fastfoward;
}

if (m_freq > 1){
    for (t = 0; t < m_freq - 1; t++){
        tst = Freq[t][1];
        m1 = tst - Freq[t][0];
        m2 = Freq[t+1][1];
        m3 = m2 - Freq[t+1][0];
        pair_each = 0;
        for (x = m1; x < tst; x++){
            t1 = Table1[x][0];
            for (y1 = m3; y1 < m2; y1++){
                t2 = Table1[y1][0];
                diff = t2 - t1;
                PI_Freq[N_PI][0] = Table1[t1][0];
                PI_Freq[N_PI][1] = Table1[t2][0];
                N_PI = N_PI + 1;
            }
        }
    }
}

85
if(diff==1||diff==2||diff==4||diff==8||diff==16||diff==32||diff==64||diff==128){
    MID[m_pair][0]=t1;
    MID[m_pair][1]=t2;
    Table_diff[m_pair][0]=diff;
    m_pair=m_pair+1;
    pair_each=pair_each+1;
    Table1[x][1]=1;
    Table1[y1][1]=1;
}
}

if (pair_each>0){
    Mid_freq[count_groups][0]=pair_each;
    count_groups=count_groups+1;
}
}

for (pzt=0;pzt<n_min;pzt++){
    if(Table1[pzt][1]!=1){
        PI[N_PI][0]=pzt;
        PI_Ndiff[N_PI]=0;
        N_PI=N_PI+1;
        P_freq=P_freq+1;
    }
}

if (count_groups==1){
    for(t=0;t<m_pair;t++){
        PI[N_PI][0]=MID[t][0];
        PI[N_PI][1]=MID[t][1];
        PI_Ndiff[N_PI]=1;
        N_PI=N_PI+1;
    }
    goto Fastfoward;
}

int tst1,tst2,tst3,tst4,tst5,i1,m11,m21,m31,x1,t31,t21,t41,t51,y2,diff_1;
int new_pair,total_pair,total_pair1;
int j3,j4;
if(count_groups>1){
    Mid_freq[0][1]=Mid_freq[0][0];
    for(t12=1;t12<count_groups;t12++){
        t14=t12-1;
        t13=Mid_freq[t14][1]+ Mid_freq[t12][0];
        Mid_freq[t12][1]=t13;
    }
    tst1=1;
    m11=count_groups-1;
    tst4=tst1*2;
    tst5=tst4*2;
    count_groups=0;
    total_pair=0;
}
    for(i1=0;i1<m11;i1++){
        new_pair=0;
        tst2=Mid_freq[i1][1];
        m21=Mid_freq[i1][1]-Mid_freq[i1][0];
        tst3=Mid_freq[i1+1][1];
        m31=Mid_freq[i1+1][1]-Mid_freq[i1+1][0];
        for(x1=m21;x1<tst2;x1++){
            t21=Table_diff[x1][0];
            for(y2=m31;y2<tst3;y2++){
if (t21==t31){
    diff_1=MID[y2][0]-MID[x1][0];
    if (diff_1==1||diff_1==2||diff_1==4||diff_1==8||diff_1==16||diff_1==32||diff_1==64||diff_1==128 ){
        FINAL[total_pair][0]=MID[x1][0];
        FINAL[total_pair][1]=MID[x1][1];
        FINAL[total_pair][2]=MID[y2][0];
        FINAL[total_pair][3]=MID[y2][1];
        FINAL_diff[total_pair][0]=t21;
        FINAL_diff[total_pair][1]=diff_1;
        total_pair=total_pair+1;
    }
}

if (new_pair>0){
    FINAL_FREQ[count_groups][0]=new_pair;
    count_groups=count_groups+1;
}

if (total_pair>0){
    total_pair1=total_pair;
    for (j4=0;j4<m_pair;j4++){
        if (CHECK[j4][0]!=1){
            PI[N_PI][0]=MID[j4][0];
            PI[N_PI][1]=MID[j4][1];
            PI_Ndiff[N_PI]=1;
            N_PI=N_PI+1;
            P_freq=P_freq+1;
        }
    }
}

int j5;
if (count_groups==1){
    for (j5=0;j5<total_pair;j5++){
        PI[N_PI][0]=FINAL[j5][0];
        PI[N_PI][1]=FINAL[j5][1];
        PI[N_PI][2]=FINAL[j5][2];
        PI[N_PI][3]=FINAL[j5][3];
        PI_Ndiff[N_PI]=2;
        N_PI=N_PI+1;
        P_freq=P_freq+1;
    }
    goto Fastfoward;
}

if (count_groups>1){
    //sort function
    for (j3=0;j3<total_pair;j3++){
        if (FINAL_diff[j3][0]>FINAL_diff[j3][1]){  
            j4=FINAL_diff[j3][0];
            FINAL_diff[j3][0]=FINAL_diff[j3][1];
            FINAL_diff[j3][1]=j4;
        }
    }
}
FINAL_diff[j3][1]=j4;
}

FINAL_FREQ[0][1]=FINAL_FREQ[0][0];
t15=FINAL_FREQ[0][1];
for(t12=1;t12<count_groups;t12++){
  t14=t12-1;
  t13=FINAL_FREQ[t14][1]+FINAL_FREQ[t12][0];
  FINAL_FREQ[t12][1]=t13;
}
tst1=2;
m11=count_groups-1;
tst4=tst1*2;
tst5=tst4*2;
count_groups=0;
total_pair=0;
for(i1=0;i1<m11;i1++){
  new_pair=0;
  tst2=FINAL_FREQ[i1][1];
  m21=FINAL_FREQ[i1][1]-FINAL_FREQ[i1][0];
  tst3=FINAL_FREQ[i1+1][1];
  m31=FINAL_FREQ[i1+1][1]-FINAL_FREQ[i1+1][0];
  for(x1=m21;x1<tst2;x1++){
    t21=FINAL_diff[x1][0];
    t31=FINAL_diff[x1][1];
    for(y2=m31;y2<tst3;y2++){
      t41=FINAL_diff[y2][0];
      t51=FINAL_diff[y2][1];
      if(t21==t41){
        if(t31==t51){
          diff_1=FINAL[x1][0]-FINAL[y2][0];
          if(diff_1==1||diff_1==2||diff_1==4||diff_1==8||diff_1==16||diff_1==32||diff_1==64||diff_1==128){
            MID[total_pair][0]=FINAL[x1][0];MID[total_pair][1]=FINAL[x1][1];MID[total_pair][2]=FINAL[x1][2];MID[total_pair][3]=FINAL[x1][3];
            MID[total_pair][4]=FINAL[y2][0];MID[total_pair][5]=FINAL[y2][1];MID[total_pair][6]=FINAL[y2][2];MID[total_pair][7]=FINAL[y2][3];
            Table_diff[y2][0]=t21;
            Table_diff[y2][0]=t31;
            Table_diff[y2][0]=diff_1;
            new_pair=new_pair+1;
            total_pair=total_pair+1;
          }
        }
      }
    }
  }
  if(new_pair>0){
    FINAL_FREQ[count_groups][0]=new_pair;
    count_groups=count_groups+1;
  }
}
if (total_pair>0){
    total_pair1=total_pair;
}
for (j4=0; j4<total_pair1; j4++){
    if (CHECK[j4][1]!=1){
        PI[N_PI][0]=FINAL[j4][0];
        PI[N_PI][1]=FINAL[j4][1];
        PI[N_PI][2]=FINAL[j4][2];
        PI[N_PI][3]=FINAL[j4][3];
        PI_Ndiff[N_PI]=2;
        N_PI=N_PI+1;
        P_freq=P_freq+1;
    }
}
if (count_groups==1){
    for (j5=0; j5<total_pair; j5++){
        PI[N_PI][0]=MID[j5][0];
        PI[N_PI][1]=MID[j5][1];
        PI[N_PI][2]=MID[j5][2];
        PI[N_PI][3]=MID[j5][3];
        PI[N_PI][4]=MID[j5][4];
        PI[N_PI][5]=MID[j5][5];
        PI[N_PI][6]=MID[j5][6];
        PI[N_PI][7]=MID[j5][7];
        PI_Ndiff[N_PI]=3;
        N_PI=N_PI+1;
        P_freq=P_freq+1;
    }
    goto Fastfoward;
}
}

//sorting

Fastfoward:
int xy,tsp,tsp1,b,temp;
//int COVER_NTERMS[9];
int *COVER_NTERMS = new int [9];
for(xy=0; xy<N_PI; xy++){
    rudia:
        tsp=PI_Ndiff[xy];
        tsp1=COVER_NTERMS[tsp];
        for(j=0; j<tsp1-1; j++){
            if (PI[xy][j]>PI[xy][j+1]){
                temp=PI[xy][j];
                PI[xy][j]=PI[xy][j+1];
                PI[xy][j+1]=temp;
            }
        }
        b=b+1;
        if (b>tsp1){
            goto rudia;
        }
}
//int PI1 [N_PI][256];
int **PI1 = new int *[N_PI];
for (count = 0; count <=(N_PI); count++)
{
    *(PI1 + count) = new (std::nothrow) int[256];
}
int M_PI,yx;
M_PI=0;
tsp=PI_Ndiff[0];
tsp1=COVER_NTERMS[tsp];
int id,jd,kd,ld,md,d_copy,tps,tps1;
for(id=0;id<N_PI;id++)
{
    d_copy=0;
    jd=id+1;
    for(kd=0;kd<4;kd++)
    {
        if(PI[id][kd]==PI[jd][kd])
        {
            d_copy=d_copy+1;
        }
    }
    if(d_copy==4){
        for(ld=jd;ld<N_PI;ld++)
        {
            tps=ld-1;
            PI[tps][0]=PI[ld][0];
            PI[tps][1]=PI[ld][1];
            PI[tps][2]=PI[ld][2];
            PI[tps][3]=PI[ld][3];
            PI_Ndiff[tps]=PI_Ndiff[ld];
        }
    }
}
for(yx=0;yx<N_PI;yx++){
    Dupli_num[yx]=PI[yx][0]+PI[yx][1]+PI[yx][2]+PI[yx][3];
}
int sum_dupli;
sum_dupli=1;
for(xy=0;xy<N_PI+1;xy++)
{
    if(Dupli_num[xy]!=Dupli_num[xy+1]){
        sum_dupli=sum_dupli+1;
    }
    else{
        sum_dupli=sum_dupli+0;
    }
}
N_PI=sum_dupli;
n_min=n_minOr;

//Developing cover chart
//int COVER_CHART[N_PI+1][n_min+1];
int **COVER_CHART = new int *[N_PI+1];
for (count = 0; count <=(N_PI+1); count++)
{
    *(COVER_CHART + count) = new (std::nothrow)

int[n_min+1];
}

//int COVERCHECK[n_min];
int *COVERCHECK = new int [n_min];

int j6,j7,j8,tpz,tpz1,j9,j10;
for(j6=0;j6<N_PI+1;j6++)
{
    for(j7=0;j7<n_min;j7++)
    {
        COVER_CHART[j6][j7]=0;
    }


```c
int NOT[N_PI];
int *NOT = new int [N_PI];

j7=0;
for(j6=0; j6<N_PI; j6++) {
    tps=PI_Ndiff[j6];
    tps1=COVER_TERMS[tps];
    NOT[j7]=tps1;
    j7=j7+1;
}

int tps2, tps3, tps4;
for(j6=0; j6<N_PI; j6++) {
    tps=NOT[j6];
    for(tps1=0; tps1<tps; tps1++) {
        tps2=PI[j6][tps1];
        for(tps3=0; tps3<n_min; tps3++) {
            tps4=MIN[tps3][0];
            if(tps2==tps4) {
                COVER_CHART[j6][tps3]=1;
            }
        }
    }
}

int N_sum;
for(j7=0; j7<n_min; j7++) {
    N_sum=0;
    for(j6=0; j6<N_PI; j6++) {
        N_sum=N_sum+COVER_CHART[j6][j7];
    }
    COVER_CHART[N_PI][j7]=N_sum;
}

//initialize min check
for(j7=0; j7<n_min; j7++) {
    COVER_CHECK[j7]=0;
}

//check each individual PI to determine essential pi
int EPI_check[N_PI];
int *EPI_check = new int [N_PI];
for(j7=0; j7<N_PI; j7++) {
    EPI_check[j7]=0;
}

for(tps=0; tps<n_min; tps++) {
    tps1=COVER_CHART[N_PI][tps];
    if(tps1==1) {
        for (j6=0; j6<N_PI; j6++) {
            if(COVER_CHART[j6][tps]==1) {
                EPI_check[j6]=1;
            }
        }
    }
}

//test if all minterms are covered
for(j9=0; j9<N_PI; j9++) {
    if(EPI_check[j9]==1) {
        j6=NOT[j9];
    }
}
```

for(j7=0; j7<n_min; j7++){
    tps=PI[j9][j7];
    for(j8=0; j8<n_min; j8++){
        if(tps==MIN[j8][0]){
            COVER_CHECK[j8]=1;
        }
    }
}

int N_sum1;
N_sum1=0;
for(j6=0; j6<n_min; j6++){
    if(COVER_CHECK[j6]==1){
        N_sum1=N_sum1+1;
    }
}

int epi;
epi=0;
// int EPI[N_PI][n_var];
int **EPI = new int *[N_PI];
for (count = 0; count <=(N_PI); count++)
{
    *(EPI + count) = new (std::nothrow) int[n_var];
}

if(N_sum1==n_min){
    for(j6=0; j6<N_PI; j6++){
        if(EPI_check[j6]==1){
            tps=PI[j6][0];
            for(j7=n_var; j7>0; j7--){
                tps1=j7-1;
                tps2=tps%2;
                EPI[epi][tps1]=tps2;
                tps=tps/2;
            }
            epi=epi+1;
        }
    }
}

//INSERTING THE DONT CARE CONDITIONS
int p_dcare,j80;
j80=0;

if(N_sum1==n_min){
    for(j6=0; j6<N_PI; j6++){
        if(EPI_check[j6]==1){
            tps=PI_Ndiff[j6];
            if(tps>0){
                tps1=2;
                tps2=tps1-1;
                for(j7=0; j7<tps; j7++){
                    p_dcare=PI[j6][tps2]-
                    PI[j6][0];
                    tps2=tps2*t2;
                    tps3=n_var-1;
                    tps4=1;
                    for(j8=0; j8<n_var; j8++){
                        if(p_dcare==tps4){
                            EPI[j80][tps3]=2;
                        }
                    }
                }
            }
        }
    }
}
tp3=tp3-1;

tps4=tps4*2;

}

j80=j80+1;

}

//CASES WHERE THE ESSENTIAL PIs do not cover all the minterms
int uncovered,uc,uc_sum;
uc=0;
if(N_sum1!=n_min){
    uncovered=n_min-N_sum1;
    //int UNC[N_PI][uncovered+1];
    int **UNC = new int *[N_PI];
    for (count = 0; count <=(N_PI); count++)
    {
        *(UNC + count) = new (std::nothrow)
    int[uncovered+1];
    }

    for(j6=0;j6<n_min;j6++){
        if(COVER_CHECK[j6]==0){
            for(j7=0;j7<N_PI;j7++){
                UNC[j7][uc]=COVER_CHART[j7][j6];
            }
            uc=uc+1;
        }
    }
    //Determining other essential prime implicants
    //First develop a table for the unchecked minterms
    //int Table[uncovered][3];
    int **Table = new int *[uncovered];
    for (count = 0; count <=(uncovered); count++)
    {
        *(Table + count) = new (std::nothrow) int[3];
    }
    int uc2;
    for(j6=0;j6<n_min;j6++){
        if(COVER_CHECK[j6]==0){
            Table[uc2][0]=MIN[j6][0];
            Table[uc2][1]=j6;
            uc2=uc2+1;
        }
    }
    //find sum of columns
    for(j6=0;j6<N_PI;j6++){
        uc_sum=0;
        for(j7=0;j7<uc; j7++){
            uc_sum=uc_sum+UNC[j6][j7];
        }
        UNC[j6][uncovered]=uc_sum;
    }
    //Find largest sum in the UNC LAST COLUMN
    int t_largest,t1;
    recheck_pi:
t_largest=UNC[0][uncovered];
t1=0;
for(j6=1;j6<N_PI;j6++){
    if(t_largest<UNC[j6][uncovered]){
        t_largest=UNC[j6][uncovered];
        t1=j6;
    }
}
EPI_check[t1]=1;
tps=NOT[t_largest];
for(j7=0;j7<tps;j7++){
    tps1=PI[t1][j7];
    for(j8=0;j8<n_min;j8++){
        if(tps1==MIN[j8][0]){  
            COVER_CHECK[j8]=1;
        }
    }
}

int N_sum2;
N_sum2=COVER_CHECK[0];
for(j6=1;j6<n_min;j6++){
    N_sum2=N_sum2+COVER_CHECK[j6];
}
if(N_sum2==n_min){
    //cout<<"confirmed\n";
    for(j6=0;j6<N_PI;j6++){  
        if(EPI_check[j6]==1){
            tps=PI[j6][0];
            for(j7=n_var;j7>0;j7--){
                tps1=j7-1;
                tps2=tps%2;
                EPI[epi][tps1]=tps2;
                tps=tps/2;
            }
            epi=epi+1;
        }
    }
}

for(j6=0;j6<N_PI;j6++){
    if(EPI_check[j6]==1){
        tps=PI_Ndiff[j6];
        if(tps>0){
            tps1=2;
            tps2=tps1-1;
            for(j7=0;j7<tps;j7++){
                p_dcare=PI[j6][tps2]-
                tps2=tps2*2;
                tps3=n_var-1;
                tps4=1;
            }
        }
    }
}
for(j8=0;j8<n_var;j8++){  
    if(p_dcare==tps4){
        EPI[j80][tps3]=2;
        tps3=tps3-1;
        tps4=tps4*2;
    }
}
}

94
j80=j80+1;
}

if(N_sum2!=n_min){
    //cout<<"say yeah\\n";
    tps=NOT[t1];
    //cout<<tps<<endl;
    for(j6=0;j6<tps;j6++)
    {
        tps1=PI[t1][j6];
        for(j7=0;j7<uncovered;j7++)
        {
            if(tps1==Table[j7][0])
            {
                for(j8=0;j8<N_PI;j8++)
                {
                    UNC[j8][j7]=0;
                }
            }
        }
    }
    goto recheck_pi;
}

//DISPLAYING THE MINIMIZED FUNCTION
//cout<<"THE FINAL REDUCED FUNCTION IS\\n";
for(j6=0;j6<epi;j6++)
{
    for(j7=0;j7<n_var;j7++)
    {
        if(EPI[j6][j7]==1){
            //String^ msg = gcnew String(lit[j7]);
            this->textBox5->Text += gcnew System::String(lit[j7]);
            this->textBox5->Text += gcnew System::String(lit[j7]);
            this->textBox5->Text += "'";
        }
    }
    if(j6<epi-1)
    {
        this->textBox5->Text += " + ";
    }
}

//IMPLEMENTATION(GENERATING LIST)
textBox1->Text += "THE IMPLEMENTATION OF THE REDUCED FUNCTION IS GIVEN BY THE NETLIST BELOW\\n";
textBox1->Text += "\\n";
textBox1->Text += "" IS SCHEMATIC DESCRIPTION FORMAT 8.0\\n";
textBox1->Text += "=====================================\\n";
textBox1->Text += "Design:  zade.pdsprj\\n";
textBox1->Text += "Doc. no.: <NONE>\\n";
textBox1->Text += "Revision: <NONE>\\n";
textBox1->Text += "Author:  <NONE>\\n";
textBox1->Text += "Created: \\
char date[9];
_strdate(date);
this->textBox1->Text += gcnew System::String(date);
textBox1->Text += "\\n";
textBox1->Text += "Modified: ");
_strdate(date);
this->textBox1->Text += gcnew System::String(date);
textBox1->Text += "\n\n"; 
textBox1->Text += "#PROPERTIES,0\n"; 
textBox1->Text += "\n"; 
textBox1->Text += "#MODELDEFS,0\n"; 
textBox1->Text += "\n"; 
textBox1->Text += "#PARTLIST,2\n"; 
textBox1->Text += "\n"; 
textBox1->Text += "#NETLIST,7\n"; 
//int IMP_TABLE[epi]; 
int *IMP_TABLE = new int [epi];
int IMP; 
IMP=0; 
for(j=0;j<epi;j++){
for(k=0;k<n_var;k++){
  if(EPI[j][k]==0||EPI[j][k]==1){ 
    IMP=IMP+1; 
  }
} 
IMP_TABLE[j]=IMP; 
IMP=0; 
} 
int T_IMP,T_IMP1,T_IMP2,T_IMP3,j_count; 
j_count=0; 
T_IMP1=0; 
T_IMP2=IMP_TABLE[0]+1; 
int j1; 
for(j=0;j<epi;j++){
  T_IMP=IMP_TABLE[j]; 
  for(k=0;k<T_IMP;k++){
    T_IMP1=T_IMP1+1; 
    textBox1->Text += T_IMP1 + ",IP,"; 
    j1=j1+1; 
    textBox1->Text += "\n\n"; 
    textBox1->Text += "U1",IP,"; 
  } 
  j1=j1+1; 
  textBox1->Text += "\n\n"; 
  textBox1->Text += "U1",IP,"; 
  textBox1->Text += "\n\n"; 
  j_count= j_count +1; 
  if (j==0){ 
    T_IMP1=T_IMP+epi; 
  } 
  j1=j1+1; 
  T_IMP2=T_IMP2+1; 
  textBox1->Text += "\n\n"; 
}
return 0; 
} 
private: System::Windows::Forms::Form^ frmMain; 
protected: 
private: System::Windows::Forms::Button^ button2; 
private: System::Windows::Forms::Button^ button1; 
private: System::Windows::Forms::TextBox^ textBox5;
private: System::Windows::Forms::Label^ label5;
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::Label^ label3;
private: System::Windows::Forms::Label^ label4;
private: System::Windows::Forms::CheckBox^ checkBox1;

private:  /// <summary>
/// Required designer variable.
/// </summary>
System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
void InitializeComponent(void)
{
    this->button2 = (gcnew System::Windows::Forms::Button());
    this->button1 = (gcnew System::Windows::Forms::Button());
    this->textBox5 = (gcnew System::Windows::Forms::TextBox());
    this->label5 = (gcnew System::Windows::Forms::Label());
    this->label2 = (gcnew System::Windows::Forms::Label());
    this->label1 = (gcnew System::Windows::Forms::Label());
    this->label3 = (gcnew System::Windows::Forms::Label());
    this->label4 = (gcnew System::Windows::Forms::Label());
    this->checkBox1 = (gcnew System::Windows::Forms::CheckBox());
    this->textBox1 = (gcnew System::Windows::Forms::TextBox());
    this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());
    this->button3 = (gcnew System::Windows::Forms::Button());
    this->textBox2 = (gcnew System::Windows::Forms::TextBox());
    this->textBox3 = (gcnew System::Windows::Forms::TextBox());
    this->textBox4 = (gcnew System::Windows::Forms::TextBox());
    this->SuspendLayout();
    // button2
    this->button2->Location = System::Drawing::Point(43, 471);
    this->button2->Name = L"button2";
    this->button2->Size = System::Drawing::Size(82, 30);
    this->button2->TabIndex = 20;
    this->button2->Text = L"<< Back";
    this->button2->UseVisualStyleBackColor = true;
    this->button2->Click += gcnew System::EventHandler(this, &FormB::button2_Click);

    // button1
    this->button1->Location = System::Drawing::Point(124, 148);
    this->button1->Name = L"button1";
    this->button1->Size = System::Drawing::Size(130, 52);
    this->button1->TabIndex = 19;
    this->button1->Text = L"Next";
    this->button1->UseVisualStyleBackColor = true;
    this->button1->Click += gcnew System::EventHandler(this, &FormB::button1_Click);

    // textBox5
    this->textBox5->Enabled = false;
this->textBox5->Location = System::Drawing::Point(473, 163);
this->textBox5->Name = L"textBox5";
this->textBox5->Size = System::Drawing::Size(286, 22);
this->textBox5->TabIndex = 17;
//
// label15
//
this->label15->AutoSize = true;
this->label15->Location = System::Drawing::Point(298, 163);
this->label15->Name = L"label15";
this->label15->Size = System::Drawing::Size(176, 17);
this->label15->TabIndex = 16;
this->label15->Text = L"Obtained reduced function";
//
// label2
//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(50, 83);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(137, 47);
this->label2->TabIndex = 15;
this->label2->Text = L"Number of minterms excluding don't care minterms";
//
// label1
//
this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(50, 25);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(137, 17);
this->label1->TabIndex = 14;
this->label1->Text = L"Number of Variables";
//
// label3
//
this->label3->Location = System::Drawing::Point(441, 81);
this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(137, 47);
this->label3->TabIndex = 22;
this->label3->Text = L"Number of don't care minterms";
//
// label4
//
this->label4->Location = System::Drawing::Point(441, 25);
this->label4->Name = L"label4";
this->label4->Size = System::Drawing::Size(137, 47);
this->label4->TabIndex = 24;
this->label4->Text = L"Are there any don't care minterms";
//
// checkBox1
//
this->checkBox1->AutoSize = true;
this->checkBox1->Location = System::Drawing::Point(677, 25);
this->checkBox1->Name = L"checkBox1";
this->checkBox1->Size = System::Drawing::Size(18, 17);
this->checkBox1->TabIndex = 25;
this->checkBox1->UseVisualStyleBackColor = true;
this->checkBox1->CheckedChanged += gcnew System::EventHandler(this, &FormB::checkBox1_CheckedChanged);
//
// text box1
//
this->textBox1->AcceptsReturn = true;
this->textBox1->BackColor = System::Drawing::SystemColors::HighlightText;
this->textBox1->Location = System::Drawing::Point(53, 206);
this->textBox1->Multiline = true;
this->textBox1->Name = L"textBox1";
this->textBox1->ReadOnly = true;
this->textBox1->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;
this->textBox1->Size = System::Drawing::Size(706, 259);
this->textBox1->TabIndex = 26;

// saveFileDialog1
this->saveFileDialog1->Filter = L"Text File|*.txt"
this->saveFileDialog1->RestoreDirectory = true;

// button3
this->button3->Location = System::Drawing::Point(660, 477);
this->button3->Name = L"button3";
this->button3->Size = System::Drawing::Size(98, 34);
this->button3->TabIndex = 27;
this->button3->Text = L"Save";
this->button3->UseVisualStyleBackColor = true;
this->button3->Click += gcnew System::EventHandler(this, &FormB::button3_Click);

// textBox2
this->textBox2->Location = System::Drawing::Point(240, 21);
this->textBox2->Name = L"textBox2";
this->textBox2->Size = System::Drawing::Size(125, 22);
this->textBox2->TabIndex = 28;
this->textBox2->TextChanged += gcnew System::EventHandler(this, &FormB::textBox2_TextChanged);

// textBox3
this->textBox3->Location = System::Drawing::Point(240, 83);
this->textBox3->Name = L"textBox3";
this->textBox3->Size = System::Drawing::Size(125, 22);
this->textBox3->TabIndex = 29;
this->textBox3->TextChanged += gcnew System::EventHandler(this, &FormB::textBox3_TextChanged);

// textBox4
this->textBox4->Location = System::Drawing::Point(623, 83);
this->textBox4->Name = L"textBox4";
this->textBox4->Size = System::Drawing::Size(125, 22);
this->textBox4->TabIndex = 30;
this->textBox4->TextChanged += gcnew System::EventHandler(this, &FormB::textBox4_TextChanged);

// FormB
this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(801, 523);
this->Controls->Add(this->textBox4);
this->Controls->Add(this->textBox3);
this->Controls->Add(this->textBox2);
this->Controls->Add(this->button3);
this->Controls->Add(this->checkBox1);
this->Controls->Add(this->label14);

99
this->Controls->Add(this->label3);
this->Controls->Add(this->button2);
this->Controls->Add(this->button1);
this->Controls->Add(this->textBox5);
this->Controls->Add(this->label15);
this->Controls->Add(this->label12);
this->Controls->Add(this->label11);
this->MaximizeBox = false;
this->Name = L"FormB";
this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;
this->Text = L"FormB";
this->ResumeLayout(false);
this->PerformLayout();
}
#pragma endregion
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
    frmMain->Show();
    this->Hide();
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    int count = 0;
    int n,i,total_min,smin,umin_limit;
    const int n_var = Convert::ToInt32(this->textBox2->Text);
    total_min=1;
    for(smin=0;smin<n_var;smin++){
        total_min=2*total_min;
    }
    int n_min = Convert::ToInt32(this->textBox3->Text);
    if(n_min<0||n_min>total_min){
        MessageBox::Show("The maximum number of minterms is"+total_min,"Error",MessageBoxButtons::OK);
        return;
    }
    int dc_num = Convert::ToInt32(this->textBox4->Text);
    //RE_DC, has to be checked before collecting anymore data
    int max_dc=n_min + dc_num;
    if(max_dc>total_min){
        //goto RE_DC;
        MessageBox::Show("Too many don't care minterms");
        return;
    }
    umin_limit=total_min-1;
    //char lit[n_var];
    char **lit = new char *[n_var];
    for (count = 0; count <=(n_var); count++)
    {
        *(lit + count) = new (std::nothrow) char[1];
    }
    for(n=0;n<n_var;n++){
        FormB1^ frmB1 = gcnew FormB1(n);
        frmB1->ShowDialog();
        //cout<<"Enter the letters you want to use to represent the final reduced function starting from MSB\n";
        lit[n][0] = frmB1->getLit();
        lit[n][1] = '\0';
    }
    //int MIN[t][2];
    int **MIN = new int *[total_min];
for (count = 0; count <= total_min; count++)
{
  *(MIN + count) = new (std::nothrow) int[2];
}
for(i=0;i<n_min;i++){
  //cout<<"Enter Minterm\n";
  FormB2^ frmB2 = gcnew FormB2(i,umin_limit);
  frmB2->ShowDialog();
  MIN[i][0] = frmB2->getMIN();
}
//Dealing with don't cares
int n_minOr, dc_trans, bol_dc;
n_minOr=n_min;
if(this->checkBox1->Checked){
bol_dc = 1;
} else{
bol_dc = 2;
}
if(bol_dc==1){
  //this->numericUpDown3->Maximum = max_dc;
  dc_num = Convert::ToInt32(this->textBox4->Text);
  max_dc = n_min + dc_num;
  //RE_DC: implemented before showing dialogs
  //int M_DC[dc_num];
  int *M_DC = new int [dc_num]; //TODO: Declare max_dc
  for(i=0;i<dc_num;i++){
    FormB3^ frmB3 = gcnew FormB3(i,umin_limit);
    frmB3->ShowDialog();
    M_DC[i] = frmB3->getMIN();
  }
  n_min = n_min + dc_num;
  dc_trans = 0;
  //TODO: M_DC[dc_trans] when dc_trans > dc_num,
  for(int c = n_minOr;c<n_min; c++){
    MIN[c][0] = M_DC[dc_trans];
    dc_trans=dc_trans+1;
  }
}

this->minWdc2(n_var,n_min,n_minOr,total_min,lit, MIN);
}
private: System::Void checkBox1_CheckedChanged(System::Object^ sender, System::EventArgs^ e) {
  if(this->checkBox1->Checked){
    this->textBox4->Enabled = true;
  } else{
    this->textBox4->Enabled = false;
  }
}
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
  saveFileDialog1->ShowDialog();
  if(saveFileDialog1->FileName != "")
  {
  
}
```csharp
String^ filename = saveFileDialog1->FileName;
StreamWriter^ file = gcnew StreamWriter(filename);
file->Write(textBox1->Text);
file->Close();
}
}
private: System::Void textBox2_TextChanged(System::Object^ sender,
System::EventArgs^ e) {
    if(textBox2->Text->Equals("")){
    }else if (!Regex::IsMatch(textBox2->Text, "^\[0-9]+$")){
        MessageBox::Show("This textbox accepts only integers", "Error", MessageBoxButtons::OK);
        String^ s = textBox2->Text;
        textBox2->Text = s->Substring(0,s->Length - 1);
        return;
    }
}
private: System::Void textBox3_TextChanged(System::Object^ sender,
System::EventArgs^ e) {
    if(textBox3->Text->Equals("")){
    }else if (!Regex::IsMatch(textBox3->Text, "^\[0-9]+$")){
        MessageBox::Show("This textbox accepts only integers", "Error", MessageBoxButtons::OK);
        String^ s = textBox3->Text;
        textBox3->Text = s->Substring(0,s->Length - 1);
        return;
    }
}
private: System::Void textBox4_TextChanged(System::Object^ sender,
System::EventArgs^ e) {
    if(textBox4->Text->Equals("")){
    }else if (!Regex::IsMatch(textBox4->Text, "^\[0-9]+$")){
        MessageBox::Show("This textbox accepts only integers", "Error", MessageBoxButtons::OK);
        String^ s = textBox4->Text;
        textBox4->Text = s->Substring(0,s->Length - 1);
        return;
    }
}
```