K MAP MINIMIZER

by

ASHNA VERMA (20BEC1083)

MARIYAM SIDDIQUI (20BEC1077)

SNEKA J (20BEC1122)

A project report submitted to

Dr. ABHINAYA S

SCHOOL OF ELECTRONICS ENGINEERING

in partial fulfilment of the requirements for the course of

CSE2003 – DATA STRUCTURES AND ALGORITHMS

in

B.TECH - Electronics and Communications Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

VELLORE INSTITUTE OF TECHNOLOGY
CHENNAI - 600127

March 2023

BONAFIDE CERTIFICATE

Certified that this project report entitled **"K-map Minimizer**" is a bonafide work of ***Ashna Verma* (20BEC1083)*, Mariyam Siddiqui* (20BEC1077)** and ***Sneka J* (20BEC1122)** who carried out the project work under my supervision and guidance for CSE2003- Data Structures and Algorithms.

Dr. ABHINAYA S

Assistant Professor

School of Electronics Engineering (SENSE),

Vellore Institute of Technology (VIT Chennai)

Chennai – 600 127

ABSTRACT

The object of solving the problem of minimizing the Boolean function in this work is a block diagram with repetition, what is the truth table of the given function. This allows to leave the minimization principle within the function calculation protocol and, thus, dispense with auxiliary objects like algebraic expressions, Karnaugh map, Veitch diagram, acyclic graph, etc.

The algebraic transformations of conjunctors are limited to the verbal form of information, they require active decoding, processing and the addition of algebraic data, therefore, as the number of variable variables increases and the resource of such minimization method is quickly exhausted. In turn, the mathematical apparatus of the combinatorial block diagram with repetition gives more information about the orthogonality, contiguity, and uniqueness of truth table blocks, so the application of such a minimization system of the Boolean function is more efficient. Equivalent transformations by graphic images, in their properties, have a large information capacity, capable of effectively replacing verbal procedures of algebraic transformations. The increased information capacity of the combinatorial method makes it possible to carry out manual minimization of 4, 5-bit Boolean functions quite easily. Using a block diagram with repetition in tasks of minimizing Boolean function is more advantageous in comparison with analogues for the following factors: – lower cost of development and implementation, since the principle of minimization of the method remains within the truth table of this function and does not require other auxiliary objects; – increasing the performance of the manual minimization procedure for 4-, 5-bit functions and increasing the performance of automated minimization with a greater number of variable functions, in particular due to the fact that several search options give the same minimum function.

The combinatorial method for minimizing Boolean functions can find practical application in the development of electronic computer systems, because: – DNF minimization is one of the multiextremal logical-combinatorial problems, the solution of which is, in particular, the combinatorial device of block-schemes with repetition; – expands the possibilities of Boolean functions minimization technology for their application in information technology; – improves the algebraic method of minimizing the Boolean function due to the tabular organization of the method and the introduction of the device of figurative numeration; – the minimum function can be obtained by several search options that reduces the complexity of the search algorithm, and is the rationale for developing a corresponding function minimization protocol.

ACKNOWLEDGEMENT

We would like to express our deep gratitude to Dr. Abhinaya S, Assistant Professor, for her patient guidance, enthusiastic encouragement, and useful critiques of the analysis.

We would also like to thank Dr. Susan Elias, Dean of School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and our respective Head of the Departments Dr. Mohanaprasad K, for his support throughout the course of our journey at VIT.

We express thanks to our parents, family, and friends for being the guiding light and providing motivation to keep us going and produce fruitful results.

ASHNA VERMA (20BEC1083)

MARIYAM SIDDIQUI (20BEC1077)

SNEKA J (20BEC1122)

1. INTRODUCTION

The Karnaugh map was invented by the American physicist Maurice Karnaugh. It is a form of logic diagram, which provides an alternative technique for representing Boolean functions. The Karnaugh map comprises a box for every line in the truth table. The binary values above the boxes are those associated with the “a” and “b” inputs. The Karnaugh map’s input values must be ordered such that the values for adjacent columns vary by only a single bit. This ordering is known as a Gray code, and it is a key factor with regard to the way in which Karnaugh maps work. Similar maps can be constructed for 3-input and 4-input functions. In the case of a 4-input map, the values associated with the c and d inputs must also be ordered as a Gray code: that is, they must be ordered in such a way that the values for adjacent rows vary by only a single bit. In the case of a 3-input Karnaugh map, any two horizontally or vertically adjacent minterms, each composed of three variables, can be combined to form a new product term composed of only two variables. Similarly, in the case of a 4-input map, any two adjacent minterms, each composed of four variables, can be combined to form a new product term composed of only three variables. When a Karnaugh map is populated using the 1s assigned to the truth table’s output, the resulting Boolean expression is extracted from the map in sum-of-products form. Karnaugh maps are most often used to represent 3-input and 4-input functions.

Karnaugh map method or K-map method is the pictorial representation of the Boolean equations and Boolean manipulations are used to reduce the complexity in solving them. These can be considered as a special or extended version of the ‘Truth table’.

Karnaugh map can be explained as “An array containing 2k cells in a grid like format, where k is the number of variables in the Boolean expression that is to be reduced or optimized”. As it is evaluated from the truth table method, each cell in the K-map will represent a single row of the truth table and a cell is represented by a square.

The cells in the k-map are arranged in such a way that there are conjunctions, which differ in a single variable, and are assigned in adjacent rows. The K-map method supports the elimination of potential race conditions and permits the rapid identification.

By using Karnaugh map technique, we can reduce the Boolean expression containing any number of variables, such as 2-variable Boolean expression, 3-variable Boolean expression, 4-variable Boolean expression and even 7-variable Boolean expressions, which are complex to solve by using regular Boolean theorems and laws.

The problems and shortcomings of the known methods for minimizing Boolean functions are associated with a rapid growth in the amount of computation, which results in an increase in the number of computational operations, and, consequently, in the increase in the number of variables of the logical function.

Simplification of Boolean expression is a practical tool to optimize programing algorithms and circuits. Several techniques have been introduced to perform the minimization, including Boolean algebra (BA), Karnaugh Map (K-Map) and QM. Minimization using BA requires high algebraic manipulation skills and will become more and more complicated when the number of terms increases. K-Map is a diagrammatic technique based on a special form of Venn diagram. It is easier to use than BA but usually it is used to handle Boolean expressions with no more than six variables. When the number of variables exceeds six, the complexity of the map is exponentially enhanced and it becomes more and more cumbersome. Functionally identical to K-Map, QM method is more executable when dealing with larger numbers of variables and is easier to be mechanized and run on a computer.

2. LITERATURE REVIEW

According to an article of International Journal of Computer Applications (0975 – 8887) Volume 15– No.7, February 2011 published by Arunachalam Solairaju Rajupillai Periyasamy, it is well known that the Karnaugh-map technique is an elegant teaching resource for academics and a systematic and powerful tool for a digital designer in minimizing low order Boolean functions.

Why is the minimization of the Boolean expression needed? By simplifying the logic function, we can reduce the original number of digital components (gates) required to implement digital circuits. Therefore, by reducing the number of gates, the chip size and the cost will be reduced and the computing speed will be increased. The K-map technique was proposed by M. Karnaugh . Later Quine and McCluskey reported tabular algorithmic techniques for the optimal Boolean function minimization. Almost all techniques have been embedded into many computer aided design packages and in all the logic design university textbooks. In the present work, a well known modeling language, the object oriented technique is used for designing an Object-oriented model for Karnaugh map with the help of digital gates. An Object-oriented algorithm is also proposed for simplification of boolean functions through K-map. The basic concepts of digital circuit design are available in many papers.

Normally a Boolean expression can be given using two forms:

1. 1 Sum-of-Products (SOP): This is the more common form of Boolean expressions. The expressions are implemented as AND gates (products) feeding a single OR gate (sum).
1. 2 Product-of-Sums (POS): This is a less commonly used form of Boolean expressions. The expressions are implemented as OR gates (sums) feeding into a single AND gate (product).

3. EXISTING WORK

In many digital circuits and practical problems we need to find expressions with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems. K-map can take two forms Sum of Product (SOP) and Product of Sum (POS) according to the need of the problem. K-map is table like representation but it gives more information than TRUTH TABLE. We fill a grid of K-map with 0's and 1's then solve it by making groups. Programming code for only 3-variable Karnaugh maps is easily available.

3.1 Steps to solve expression using K-map:

1. Select K-map according to the number of variables.
2. Identify minterms or maxterms as given in the problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).
4. For POS put 0's in blocks of K-map respective to the maxterms(1's elsewhere).
5. Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as you can in one group.
6. From the groups made in step 5 find the product terms and sum them up for SOP form.

3.2 Proposed Work:

We proposed a programming code for the minimization of n-variable Karnaugh maps using the Quine-Mccluskey algorithm and Petrick's method.

Steps for our approach:

1. Separation all the minterms (and don't cares) of a function into groups
2. Merge minterms from adjacent groups to form a new implicant table
3. Repeat step 2 until no more merging is possible
4. Put all prime implicants in a cover table (don't cares excluded)

5. Recognize fundamental minterms, and henceforth basic prime implicants
6. Add prime implicants to the minimum expression of F until all minterms of F are covered.
7. So after simplification through the QM method, a minimum expression for the function F is found.

4. WORKING MODULES AND FLOW CHART/ SYSTEM DESIGN OF PROPOSED WORK

- Divide all the minterms (and don't cares) of a function into groups
- Merge minterms from adjacent groups to form a new implicant table
- Repeat step 2 until no more merging is possible
- Put all prime implicants in a cover table (don't cares excluded)
- Identify essential minterms, and hence essential prime implicants
- Add prime implicants to the minimum expression of F until all minterms of F are covered.
- So after simplification through the QM method, a minimum expression for the function F is found.

5. FLOWCHART

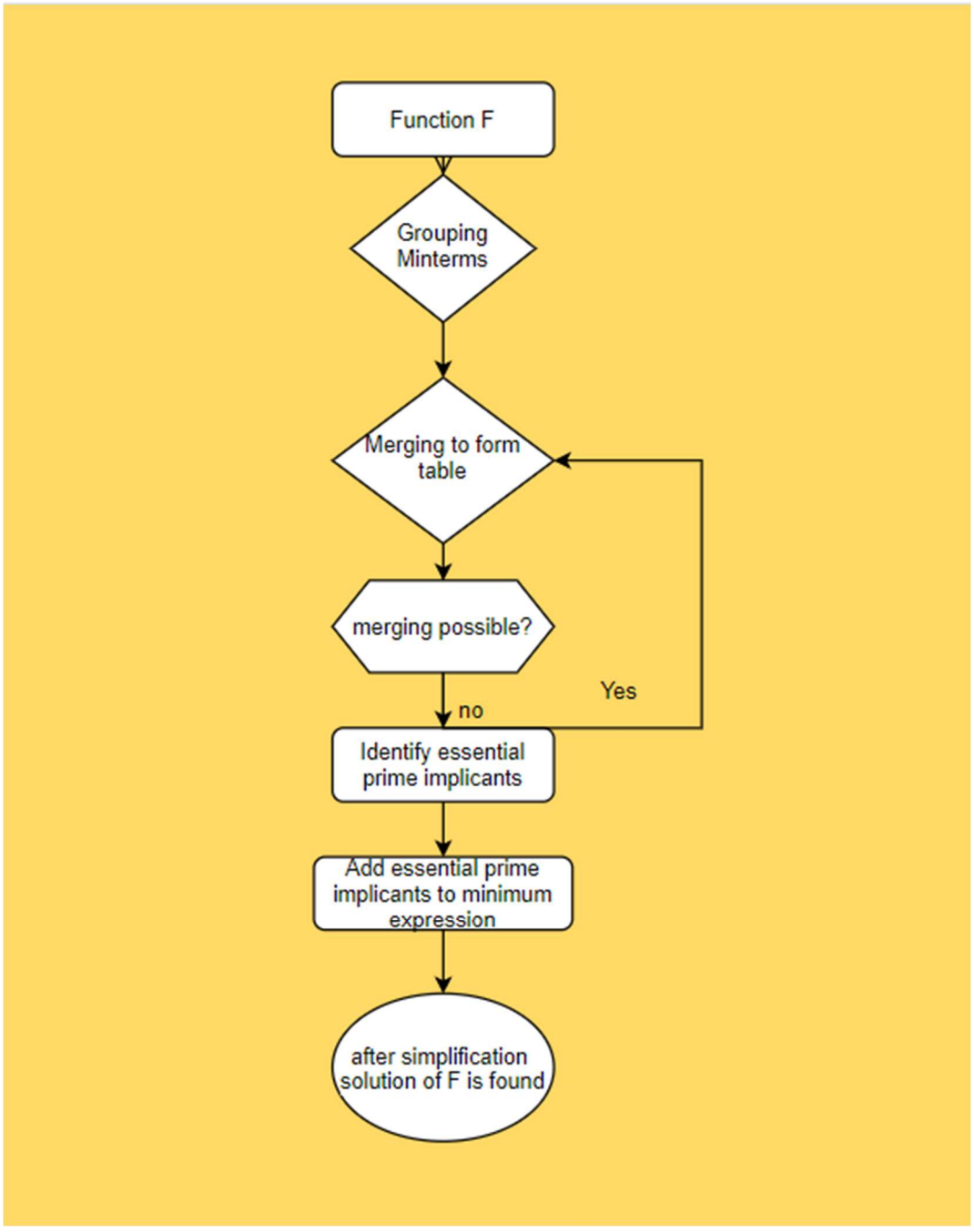
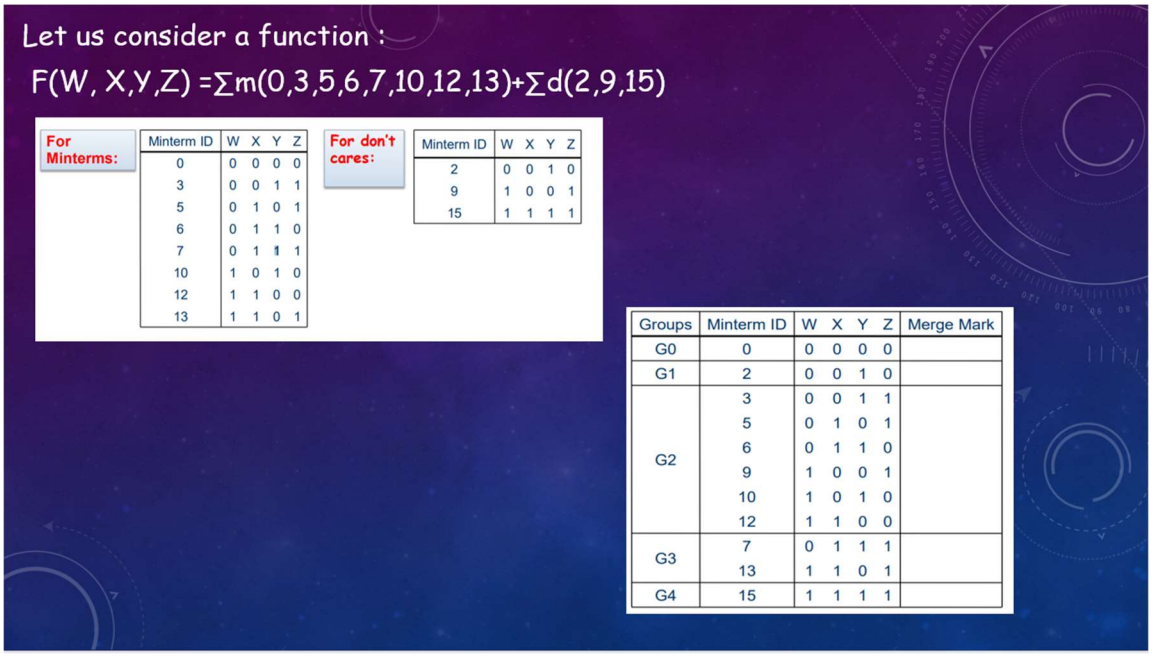


Figure 1:Flow chart of quine Mc cluskey algorithm used

6. DETAILED WORK

6.1 Description of each modules with appropriate diagram and explanation :

1. Grouping minterms:



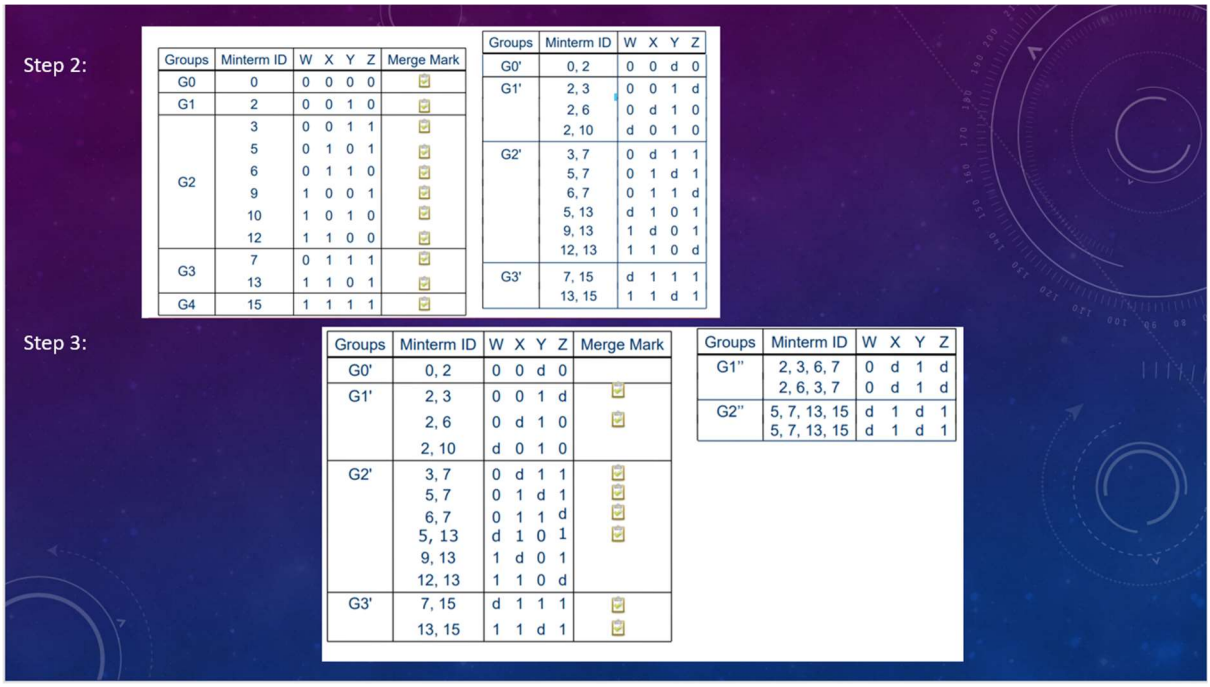
Minterm ID	W	X	Y	Z
0	0	0	0	0
3	0	0	1	1
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
10	1	0	1	0
12	1	1	0	0
13	1	1	0	1

Minterm ID	W	X	Y	Z
2	0	0	1	0
9	1	0	0	1
15	1	1	1	1

Groups	Minterm ID	W	X	Y	Z	Merge Mark
G0	0	0	0	0	0	
G1	2	0	0	1	0	
G2	3	0	0	1	1	
	5	0	1	0	1	
	6	0	1	1	0	
	9	1	0	0	1	
	10	1	0	1	0	
	12	1	1	0	0	
G3	7	0	1	1	1	
	13	1	1	0	1	
G4	15	1	1	1	1	

Grouping the minterms according to the no.of 1's present in its binary form

2. Merging minterms:



Groups	Minterm ID	W	X	Y	Z	Merge Mark
G0	0	0	0	0	0	✓
G1	2	0	0	1	0	✓
G2	3	0	0	1	1	✓
	5	0	1	0	1	✓
	6	0	1	1	0	✓
	9	1	0	0	1	✓
	10	1	0	1	0	✓
	12	1	1	0	0	✓
G3	7	0	1	1	1	✓
	13	1	1	0	1	✓
G4	15	1	1	1	1	✓

Groups	Minterm ID	W	X	Y	Z
G0'	0, 2	0	0	d	0
G1'	2, 3	0	0	1	d
	2, 6	0	d	1	0
	2, 10	d	0	1	0
G2'	3, 7	0	d	1	1
	5, 7	0	1	d	1
	6, 7	0	1	1	d
	5, 13	d	1	0	1
	9, 13	1	d	0	1
	12, 13	1	1	0	d
G3'	7, 15	d	1	1	1
	13, 15	1	1	d	1

Groups	Minterm ID	W	X	Y	Z	Merge Mark
G0'	0, 2	0	0	d	0	
G1'	2, 3	0	0	1	d	✓
	2, 6	0	d	1	0	✓
	2, 10	d	0	1	0	
G2'	3, 7	0	d	1	1	✓
	5, 7	0	1	d	1	✓
	6, 7	0	1	1	d	✓
	5, 13	d	1	0	1	✓
	9, 13	1	d	0	1	
	12, 13	1	1	0	d	
G3'	7, 15	d	1	1	1	✓
	13, 15	1	1	d	1	✓

Groups	Minterm ID	W	X	Y	Z
G1''	2, 3, 6, 7	0	d	1	d
	2, 6, 3, 7	0	d	1	d
G2''	5, 7, 13, 15	d	1	d	1
	5, 7, 13, 15	d	1	d	1

Merging those minterms till no more merging is possible

3. Identifying Prime implicants and forming cover table:

Groups	Minterm ID	W	X	Y	Z	Merge Mark
G0"	0, 2	0	0	d	0	
G1"	2, 3, 6, 7	0	d	1	d	
	2, 10	d	0	1	0	
G2"	5, 7, 13, 15	d	1	d	1	
	9, 13	1	d	0	1	
	12, 13	1	1	0	d	

No more merging possible!

- Step 4:

Minterm ID	$\overline{W}\,\overline{X}\,\overline{Z}$	$\overline{W}Y$	$\overline{X}Y\overline{Z}$	XZ	$WX\overline{Y}$	$W\overline{Y}Z$
0	1					
3		1				
5				1		
6		1				
7		1		1		
10			1			
12					1	
13				1	1	1

Identifying prime implicants to form a cover table

4. Adding implicants to the minimum expression and finding a solution.

- Step 5:

Minterm ID	$\overline{W}\,\overline{X}\,\overline{Z}$	$\overline{W}Y$	$\overline{X}Y\overline{Z}$	XZ	$WX\overline{Y}$	$W\overline{Y}Z$
0	[1]					
3		[1]				
5				[1]		
6		[1]				
7		1		1		
10			[1]			
12					[1]	
13				1	1	1

Already covered all minterms!

Therefore , minimum expression for F(W, X, Y, Z) is: F(W, X ,Y, Z) =W' X' Z' +W'Y + X'Y Z' + XZ +WX Y'

Adding the essential implicants to the minimum expression till the function is covered and hence solution is obtained by simplifying.

7. CODE

```
class Minterm:
    def __init__(self, value, dont_cares):
        self.value = value
        self.dont_cares = dont_cares
        self.group = None
        self.next = None

class Group:
    def __init__(self, terms):
        self.terms = terms
        self.next = None

def compare_minterms(m1, m2):
    num_diff_bits = 0
    for i in range(len(m1.value)):
        if m1.value[i] != m2.value[i] and (m1.dont_cares[i] == '0' or m2.dont_cares[i] == '0'):
            num_diff_bits += 1
            if num_diff_bits > 1:
                return False
    return True

def generate_pairs(minterms):
    pairs = []
    for i in range(len(minterms)):
        for j in range(i + 1, len(minterms)):
            if compare_minterms(minterms[i], minterms[j]):
                pairs.append((minterms[i], minterms[j]))
    return pairs

def generate_groups(minterms):
```

```
    groups = {}
    for minterm in minterms:
        if minterm.dont_cares.count('0') not in groups:
            groups[minterm.dont_cares.count('0')] = Group([minterm])
        else:
            groups[minterm.dont_cares.count('0')].terms.append(minterm)
    return groups

def combine_groups(groups):
    new_groups = {}
    for i in range(len(groups) - 1):
        for j in range(i + 1, len(groups)):
            if abs(len(groups[i].terms) - len(groups[j].terms)) == 1:
                combined_terms = []
                for term1 in groups[i].terms:
                    for term2 in groups[j].terms:
                        if compare_minterms(term1, term2):
                            combined_terms.append(Minterm(term1.value, term1.dont_cares))
                            break
                    else:
                        continue
                    break
                else:
                    continue
                new_groups[groups[i], groups[j]] = Group(combined_terms)
    return new_groups

def eliminate_redundant_groups(groups):
    for group1 in groups.values():
        for group2 in groups.values():
            if group1 is not group2 and set(group1.terms).issubset(set(group2.terms)):
                group2.terms = [term for term in group2.terms if term not in group1.terms]
```

```
def minimize_function(minterms):
    linked_list = [Minterm(minterm, '-'*len(minterm)) for minterm in minterms]
    groups = generate_groups(linked_list)
    while True:
        new_groups = combine_groups(groups.values())
        if not new_groups:
            break
        groups.update(new_groups)
    eliminate_redundant_groups(groups)
    return [group.terms for group in groups.values()]



def mul(x,y): # Multiply 2 minterms
    res = []
    for i in x:
        if i+"'" in y or (len(i)==2 and i[0] in y):
            return []
        else:
            res.append(i)
    for i in y:
        if i not in res:
            res.append(i)
    return res

def multiply(x,y): # Multiply 2 expressions
    res = []
    for i in x:
        for j in y:
            tmp = mul(i,j)
            res.append(tmp) if len(tmp) != 0 else None
    return res

def refine(my_list,dc_list): # Removes don't care terms from a given list and returns refined list
```

```
    res = []
    for i in my_list:
        if int(i) not in dc_list:
            res.append(i)
    return res

def findEPI(x): # Function to find essential prime implicants from prime implicants chart
    res = []
    for i in x:
        if len(x[i]) == 1:
            res.append(x[i][0]) if x[i][0] not in res else None
    return res

def findVariables(x): # Function to find variables in a meanterm. For example, the minterm --01 has C' and D as variables
    var_list = []
    for i in range(len(x)):
        if x[i] == '0':
            var_list.append(chr(i+65)+"'")
        elif x[i] == '1':
            var_list.append(chr(i+65))
    return var_list

def flatten(x): # Flattens a list
    flattened_items = []
    for i in x:
        flattened_items.extend(x[i])
    return flattened_items

def findminterms(a): #Function for finding out which minterms are merged. For example, 10-1 is obtained by merging 9(1001) and 11(1011)
    gaps = a.count('-')
    if gaps == 0:
```

```
        return [str(int(a,2))]
    x = [bin(i)[2:].zfill(gaps) for i in range(pow(2,gaps))]
    temp = []
    for i in range(pow(2,gaps)):
        temp2,ind = a[:],-1
        for j in x[0]:
            if ind != -1:
                ind = ind+temp2[ind+1:].find('-')+1
            else:
                ind = temp2[ind+1:].find('-')
            temp2 = temp2[:ind]+j+temp2[ind+1:]
        temp.append(str(int(temp2,2)))
        x.pop(0)
    return temp

def compare(a,b): # Function for checking if 2 minterms differ by 1 bit only
    c = 0
    for i in range(len(a)):
        if a[i] != b[i]:
            mismatch_index = i
            c += 1
            if c>1:
                return (False,None)
    return (True,mismatch_index)

def removeTerms(_chart,terms): # Removes minterms which are already covered from chart
    for i in terms:
        for j in findminterms(i):
            try:
                del _chart[j]
            except KeyError:
                pass
```

```
mt = [int(i) for i in input("Enter the minterms: ").strip().split()]
dc = [int(i) for i in input("Enter the don't cares(If any): ").strip().split()]
mt.sort()
minterms = mt+dc
minterms.sort()
size = len(bin(minterms[-1]))-2
groups,all_pi = {},set()

# Primary grouping starts
for minterm in minterms:
    try:
        groups[bin(minterm).count('1')].append(bin(minterm)[2:].zfill(size))
    except KeyError:
        groups[bin(minterm).count('1')] = [bin(minterm)[2:].zfill(size)]
# Primary grouping ends

#Primary group printing starts
print("\n\n\n\nGroup No.\tMinterms\tBinary of Minterms\n%s"%('='*50))
for i in sorted(groups.keys()):
    print("%5d:"%i) # Prints group number
    for j in groups[i]:
        print("\t\t    %-20d%s"%(int(j,2),j)) # Prints minterm and its binary representation
    print('-'*50)
#Primary group printing ends

# Process for creating tables and finding prime implicants starts
while True:
    tmp = groups.copy()
    groups,m,marked,should_stop = {},0,set(),True
    l = sorted(list(tmp.keys()))
    for i in range(len(l)-1):
        for j in tmp[l[i]]: # Loop which iterates through current group elements
            for k in tmp[l[i+1]]: # Loop which iterates through next group elements
```

```
            res = compare(j,k) # Compare the minterms
            if res[0]: # If the minterms differ by 1 bit only
                try:
                    groups[m].append(j[:res[1]]+'-'+j[res[1]+1:]) if j[:res[1]]+'-'+j[res[1]+1:] not in groups[m]
else None # Put a '-' in the changing bit and add it to corresponding group
                except KeyError:
                    groups[m] = [j[:res[1]]+'-'+j[res[1]+1:]] # If the group doesn't exist, create the group at first
and then put a '-' in the changing bit and add it to the newly created group
                should_stop = False
                marked.add(j) # Mark element j
                marked.add(k) # Mark element k
        m += 1
    local_unmarked = set(flatten(tmp)).difference(marked) # Unmarked elements of each table
    all_pi = all_pi.union(local_unmarked) # Adding Prime Implicants to global list
    print("Unmarked elements(Prime Implicants) of this table:",None if len(local_unmarked)==0 else ',
'.join(local_unmarked)) # Printing Prime Implicants of current table
    if should_stop: # If the minterms cannot be combined further
        print("\n\nAll Prime Implicants: ",None if len(all_pi)==0 else ', '.join(all_pi)) # Print all prime
implicants
        break
    # Printing of all the next groups starts
    print("\n\n\n\nGroup No.\tMinterms\tBinary of Minterms\n%s"%('='*50))
    for i in sorted(groups.keys()):
        print("%5d:"%i) # Prints group number
        for j in groups[i]:
            print("\t\t%-24s%s"%(','.join(findminterms(j)),j)) # Prints minterms and its binary representation
        print('-'*50)
    # Printing of all the next groups ends
# Process for creating tables and finding prime implicants ends


# Printing and processing of Prime Implicant chart starts
sz = len(str(mt[-1])) # The number of digits of the largest minterm
```

```
chart = {}
print('\n\n\nPrime Implicants chart:\n\n    Minterms    |%s\n%s'%(' '.join((' '*(sz-len(str(i))))+str(i) for i in
mt),'='*(len(mt)*(sz+1)+16)))
for i in all_pi:
    merged_minterms,y = findminterms(i),0
    print("%-16s|"%','.join(merged_minterms),end='')
    for j in refine(merged_minterms,dc):
        x = mt.index(int(j))*(sz+1) # The position where we should put 'X'
        print(' '*abs(x-y)+' '*(sz-1)+'X',end='')
        y = x+sz
        try:
            chart[j].append(i) if i not in chart[j] else None # Add minterm in chart
        except KeyError:
            chart[j] = [i]
    print('\n'+'-'*(len(mt)*(sz+1)+16))
# Printing and processing of Prime Implicant chart ends

EPI = findEPI(chart) # Finding essential prime implicants
print("\nEssential Prime Implicants: "+', '.join(str(i) for i in EPI))
removeTerms(chart,EPI) # Remove EPI related columns from chart

if(len(chart) == 0): # If no minterms remain after removing EPI related columns
    final_result = [findVariables(i) for i in EPI] # Final result with only EPIs
else: # Else follow Petrick's method for further simplification
    P = [[findVariables(j) for j in chart[i]] for i in chart]
    while len(P)>1: # Keep multiplying until we get the SOP form of P
        P[1] = multiply(P[0],P[1])
        P.pop(0)
    final_result = [min(P[0],key=len)] # Choosing the term with minimum variables from P
    final_result.extend(findVariables(i) for i in EPI) # Adding the EPIs to final solution
print('\n\nSolution: F = '+ '.join(''.join(i) for i in final_result))

input("\nPress enter to exit...")
```

8. OUTPUTS AND OBSERVATIONS

SAMPLE INPUT:

F(P,Q,R,S)=∑(0,3,5,6,7,10,12,13)

Don't cares= ∑(2,9,15)

```
PS C:\Users\Ashna Verma\Desktop\C C++> python -u "c:\Users\Ashna Verma\Desktop\C C++\test2.py"
Enter the minterms: 0 3 5 6 7 10 12 13
Enter the don't cares(If any): 2 9 15



Group No.       Minterms        Binary of Minterms
================================================
    0:
                0                   0000
------------------------------------------------
    1:
                2                   0010
------------------------------------------------
    2:
                3                   0011
                5                   0101
                6                   0110
                9                   1001
                10                  1010
                12                  1100
------------------------------------------------
    3:
                7                   0111
                13                  1101
------------------------------------------------
    4:
                15                  1111
------------------------------------------------
Unmarked elements(Prime Implicants) of this table: None
```

```
Group No.       Minterms        Binary of Minterms
================================================
    0:
                0,2                     00-0
------------------------------------------------
    1:
                2,3                     001-
                2,6                     0-10
                2,10                    -010
------------------------------------------------
    2:
                3,7                     0-11
                5,7                     01-1
                5,13                    -101
                6,7                     011-
                9,13                    1-01
                12,13                   110-
------------------------------------------------
    3:
                7,15                    -111
                13,15                   11-1
------------------------------------------------
Unmarked elements(Prime Implicants) of this table: 1-01, -010, 00-0, 110-
```

```
Group No.       Minterms        Binary of Minterms
================================================
    1:
                2,3,6,7                 0-1-
------------------------------------------------
    2:
                5,7,13,15               -1-1
------------------------------------------------
Unmarked elements(Prime Implicants) of this table: 0-1-, -1-1


All Prime Implicants:  1-01, 0-1-, -1-1, -010, 00-0, 110-



Prime Implicants chart:

    Minterms    | 0  3  5  6  7 10 12 13
=====================================
9,13            |                    X
-------------------------------------
2,3,6,7         |    X     X  X
-------------------------------------
5,7,13,15       |       X     X        X
-------------------------------------
2,10            |               X
-------------------------------------
0,2             | X
-------------------------------------
12,13           |                  X  X
-------------------------------------

Essential Prime Implicants: 0-1-, -1-1, -010, 00-0, 110-
```

```
Solution: F = A'C + BD + B'CD' + A'B'D' + ABC'

Press enter to exit...
```

9. CONCLUSION

The Karnaugh Map Minimizer project is a useful tool for reducing logical expressions and simplifying the design of digital circuits. The program implements a step-by-step algorithm for minimizing Boolean expressions using Karnaugh maps, and it allows the user to input the Boolean expression or the truth table directly. The program outputs the simplified expression, which can be used to optimize the digital circuit design.

The Karnaugh Map Minimizer project demonstrates the power of Boolean algebra and the usefulness of Karnaugh maps in simplifying digital circuits. The implementation of the algorithm and the user-friendly interface make it easy for anyone to use the program to simplify their logical expressions. Overall, this project is a valuable tool for digital circuit designers and students learning digital design.

10. FUTURE WORK

Future scopes for the K-map minimizer project:

1. GUI implementation: Currently, the K-map minimizer operates through a command-line interface. In the future, a graphical user interface (GUI) could be developed to make it more user-friendly and accessible to users who are less comfortable with command-line tools.

2. Optimization algorithms: While the current implementation of the K-map minimizer is effective, there may be additional optimization algorithms that could be implemented to further improve its efficiency and accuracy.

3. Integration with other tools: The K-map minimizer could be integrated with other tools used in digital logic design to create a more comprehensive suite of software for designing and optimizing digital circuits.

4. Cloud-based implementation: A cloud-based implementation of the K-map minimizer could allow users to access the software from anywhere with an internet connection, making it easier to collaborate on circuit designs.

5. Support for additional logic families: Currently, the K-map minimizer supports the Boolean logic family. In the future, support could be added for other logic families, such as ternary or fuzzy logic.

Overall, We have done this project to solve in SOP form , in the future it can be implemented to solve the functions in POS form. The K-map minimizer project has a lot of potential for future development and improvement. By continuing to refine the software and add new features, it could become an even more valuable tool for digital logic designers.

11. REFERENCES

According to an article published by Scientia Iranica Volume 19, Issue 3, June 2012, Pages 690-695**,** Simplified schemes for generation of error correcting codes, using Karnaugh map, are being proposed. Also, analytical work done by Muller [2] and Reed [3] has had significant contributions to the field. Reviriego et al. have done an interesting research in soft error detection and correction [4]. Payandeh et al. have considerable contributions to secure channel coding [5]. In [6], Biberstein and Etzion have achieved a profound theoretical research on single error correction and double-adjacent-error detection. Helgert and Stinaff have approached the problem of correcting codes by using tables of minimum distance between them [7]. Also, deep theoretical work has been done on the subject and books published [8], [9], [10]. However, simpler handling of the Karnaugh map made us choose this technique that by visualizing the error positions in the map, offers remarkable ease in code generation and also understanding. As a result of the Gray code arrangement in the rows and columns, we can use visual reasoning rather than having to go through mathematical concepts and sometimes complex definitions and proofs leading to the same results. According to an article published by Visvam Devadoss Ambeth Kumar and S.Gokul Amuthan The disadvantage of formal K-map is that it can solve up to 6 variable only. This disadvantage can be recovered by using DC- K-Map.

This is similar to formal K-map but it can be used to solve up to n-variable. It requires only the knowledge of mapping a 4 variable K-map. As it uses mapping technique up to 4 variable it's easy to map the map the cells efficiently.

- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.3695&rep=rep1&type=pdf
- https://pdfs.semanticscholar.org/e73d/22864f5c4b6505dda37a2ad65979e86c87a0.pdf
- https://www.geeksforgeeks.org/introduction-of-k-map-karnaugh-map/
- https://en.wikipedia.org/wiki/Quine%E2%80%93McCluskey_algorithm

- /www.sciencedirect.com/science/article/pii/S1026309811001805